



PIE Tech

POLLACHI INSTITUTE OF ENGINEERING AND TECHNOLOGY
(Approved by *AICTE* and Affiliated to *Anna University*) *sky is the limit*

Department of computer science and engineering

Regulation 2021

III Year- V Semester

CS3551 DISTRIBUTED COMPUTING

CS3551 DISTRIBUTED COMPUTING

UNIT I INTRODUCTION

Introduction: Definition-Relation to Computer System Components – Motivation – Message -Passing Systems versus Shared Memory Systems – Primitives for Distributed Communication – Synchronous versus Asynchronous Executions – Design Issues and Challenges; A Model of Distributed Computations: A Distributed Program – A Model of Distributed Executions – Models of Communication Networks – Global State of a Distributed System.

UNIT II LOGICAL TIME AND GLOBAL STATE

Logical Time: Physical Clock Synchronization: NTP – A Framework for a System of Logical Clocks – Scalar Time – Vector Time; Message Ordering and Group Communication: Message Ordering Paradigms – Asynchronous Execution with Synchronous Communication – Synchronous Program Order on Asynchronous System – Group Communication – Causal Order – Total Order; Global State and Snapshot Recording Algorithms: Introduction – System Model and Definitions – Snapshot Algorithms for FIFO Channels.

UNIT III DISTRIBUTED MUTEX AND DEADLOCK

Distributed Mutual exclusion Algorithms: Introduction – Preliminaries – Lamport’s algorithm – RicartAgrawala’s Algorithm — Token-Based Algorithms – Suzuki-Kasami’s Broadcast Algorithm; Deadlock Detection in Distributed Systems: Introduction – System Model – Preliminaries – Models of Deadlocks – Chandy-Misra-Haas Algorithm for the AND model and OR Model

UNIT IV CONSENSUS AND RECOVERY

Consensus and Agreement Algorithms: Problem Definition – Overview of Results – Agreement in a Failure-Free System(Synchronous and Asynchronous) – Agreement in Synchronous Systems with Failures; Checkpointing and Rollback Recovery: Introduction – Background and Definitions – Issues in Failure Recovery – Checkpoint-based Recovery – Coordinated Checkpointing Algorithm - - Algorithm for Asynchronous Checkpointing and Recovery

UNIT V CLOUD COMPUTING

Definition of Cloud Computing – Characteristics of Cloud – Cloud Deployment Models – Cloud Service Models – Driving Factors and Challenges of Cloud – Virtualization – Load Balancing – Scalability and Elasticity – Replication – Monitoring – Cloud Services and Platforms: Compute Services – Storage Services – Application Services

(X) 2 Marks

Parallel Computing

(*) The goal of parallel computing has traditionally been to provide performance, either in terms of processor power or memory.

(*) Interaction b/w processors is frequent.

(*) It is typically fine grained with low overhead.

(*) Assumed to be reliable.

(*) Pc values short execution time

Distributed Computing

(*) The goal is to provide Convenience, where Convenience includes high availability, reliability & physical distribution

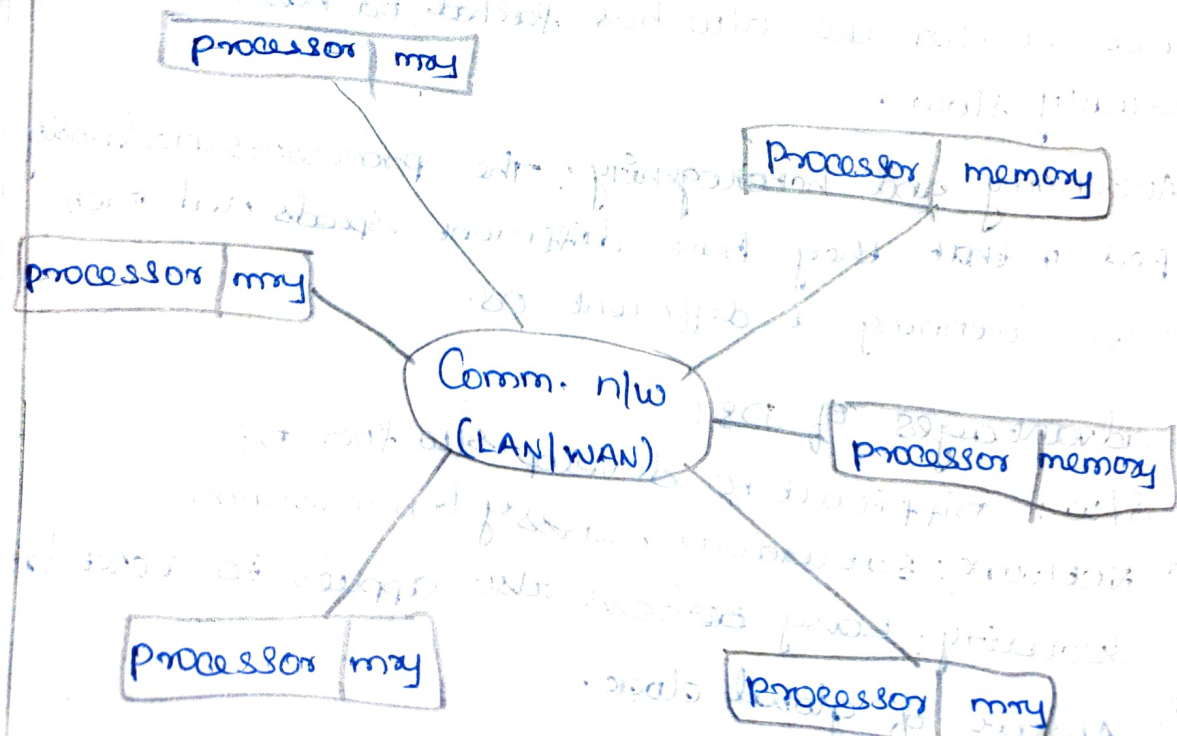
(*) Interaction is infrequent.

(*) It is heavier weight

(*) Assumed to be unreliable

(*) Dc values long up time

Relation to Computer s/m Components:



⇒ In DC sim, each node consists of a processor (cpu), local memory and interface. Communication b/w any two or more nodes is only by message passing because there is no common mem available.

⇒ Also termed as middleware. DC uses a layered architecture to break down the complexity of sim design.

⇒ A DS can consist of any number of possible configurations, such as mainframes, personal computers, workstations, mini computers and so on.

Motivation:

- ① Economies: A collection of microprocessors offer a better price/perfor than mainframes. Low price/perfor ratio is the cost effective way to increase computing power.
- ② Speed: A DS may have more total computing power than a mainframe.
- ③ DS can be extended through the addition of components, thereby providing better scalability, compared to centralized sims.
- ④ Inherent distribution: Some applications are inherently distributed, eg: a supermarket chain.
- ⑤ Reliability: If one sim crashes, the sim as a whole can still survive.
- ⑥ Incremental growth: Computing power can be added in small increments.

1.2) Need of Distributed System:

- (*) Resource sharing is main motivation of ds.
- (*) Resource may be s/w resource or h/w resource
⇒ printers, disks, CDROM, data, files, database
- (*) Primary requirements of DS:
 - ① Fault tolerance
 - ② Consistency of replicated data
 - ③ Security
 - ④ Reliability
 - ⑤ Concurrent transaction

1.3) Focus of Resource sharing:

Types of resources:

- ① H/w resource: Hard disk, printer, camera, scanner
- ② Data: Files, database, web pages
- ③ Service: Search Engine

Patterns of resource sharing vary widely in their scope and how closely users work together.

- ① Search Engine: users need no contact b/w users.
- ② Computer supported Co-operative working (CSCW): users cooperate directly share resources.

Service: manages a collection of related resources and presents their functionalities to user & applications.

Server is basically storage of resources & provides services to the authenticated clients.

The complete interaction b/w server machine & client machine, from the point when the client sends it request to when it receives the server's response, is called remote invocation.

H/W resources:

- ① CPU: a) Computing server
b) Remote object server
c) Worm prgm.
- ② memory: Cache server holds recently accessed webpage in its RAM.
- ③ Disk: File server, virtual disk server.
- ④ screen: N/w window s/ms, (x-11)
- ⑤ Printer: queuing s/m. from many computers.

S/w resources:

- ① web page:
- ② file:
- ③ object:
- ④ Database:
- ⑤ Newsgroup contents:
- ⑥ video/audio stream:

13 Mark

Message passing s/m vs Shared mem s/ms:

Msg passing: Comm. with each other by passing msgs.

⇒ Direct Comm:

⇒ Indirect Comm. → uses mailbox for sending & receiving msg.

(*) Fixed & variable size

a) send (dest-name, msg)

b) receive (source-name, msg)

Message passing s/ms

Shared memory s/ms.

1. platforms that exchange messaging for sharing data are called msg passing platforms.

platforms that provide shared mem for data share are called multiprocessors.

2. Msg passing is useful for sharing small amounts of data so that conflicts need not occur.

It makes sure that the processors are not writing the same location simultaneously.

3. The comm. is slower when compared to shared mem techniques.

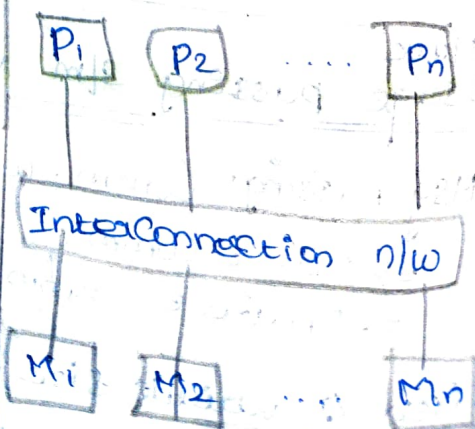
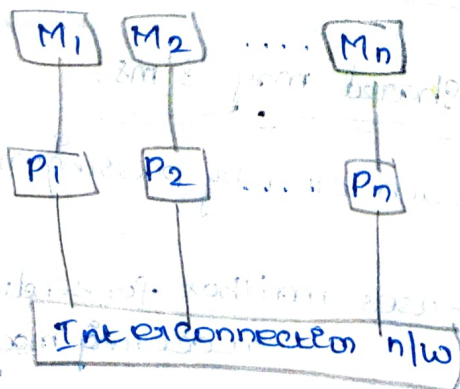
It is faster comm. strategy when compared to msg passing technique.

4. In this process model, the processes communicate with others by exchanging msg's.

If the process wishes to initiate comm. and has to share, create a shared mem region in its address space.

5. This techniques can be used in heterogeneous computers

cannot be used to heterogeneous computers.



Primitives for Distributed Communication:

Blocking / Non-blocking , Synchronous / Asynchronous

message send comm primitives is denoted by
`Send()`

msg receive comm primitives
`Receive()`

msg passing Primitive Commands:

`SEND (msg, dest)`

`RECEIVE (src, buffer)`

It uses two options for sending data:

Buffered - User data is copied in the kernel buffer.

Unbuffered - The data gets copied directly from the user buffer onto the n/w.

Three combinations are possible using blocking & non blocking

a) Blocking send, blocking receive:

⇒ Both sender & receiver are blocked until the msg is delivered. This is called Rendezvous.

⇒ Tight synchronization b/w processes.

b) Nonblocking send, blocking receive:

Sender may continue on, the receiver is blocked until the requested msg arrives.

c) Nonblocking send, nonblocking receive:

Sending process sends the msg and resumes the operation. receiver retrieves either a valid msg or a null. (i.e) neither party is required to wait.

Q. 24/26

Synchronous Vs Asynchronous executions:

Synchronous execution

Asynchronous execution

1. Means the first task in a pgm must finish processing before moving on to executing the next task.

Means a second task can begin executing in parallel, without waiting for an earlier task to finish.

2. Lower and upper bounds on execution time of processes can be set.

No bound on process exe. time.

3. Transmitted msgs are received within a known bounded time.

No bound on msg transmission delay.

4. Drift rates b/w local clocks have a known bound.

No bounds on drift rates b/w local clocks.

Design issues and challenges:

Challenges from s/m perspective: — [N/D-202]

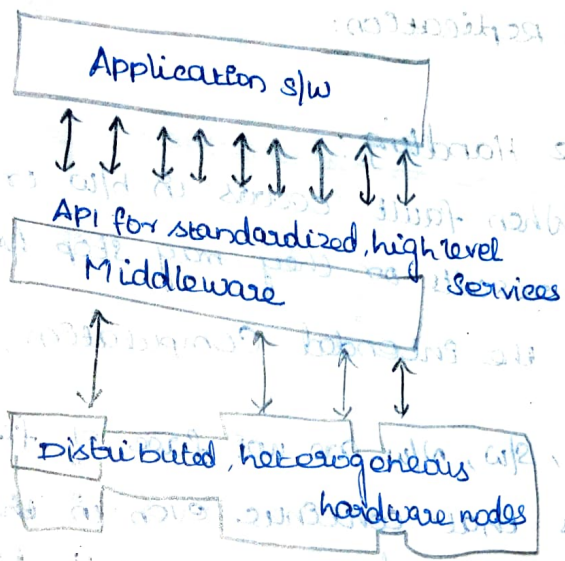
- ① Communication Mechanisms:
- ② Processes:
- ③ Naming:
- ④ Synchronization:
- ⑤ Data storage and access:
- ⑥ Consistency & replication:
- ⑦ Distributed S/m security:

Challenges

1) Heterogeneity - Highly heterogeneous in many dimensions, including bandwidth, processor, speed, disk capacity, security, failure rate.

- 1) Computer N/w :
- 2) Computer H/w devices:
- 3) operating s/ms :
- 4) pgming languages:
- 5) Different roles of s/w developers, designers, & m mng

Middleware:



2) Openness - s/m can be easily extended and modified. New Components can be integrated with existing Components.

3) Security : - more important in DS.

Three Components:

- ① Confidentiality
- ② Integrity
- ③ Availability

4) Scalability: - Can handle the addition of users and resources without suffering a noticeable loss of performance.

- ① In size:
- ② In location:
- ③ In administration:

Scaling Techniques:

- ① Handling Comm. Latencies
- ② Distribution:
- ③ Replication:

5) Failure Handling:

When fault occurs in h/w or s/w, pgms may produce results or they may stop before they have completed the intended computation.

H/w, s/w, n/w, are not free of failures.

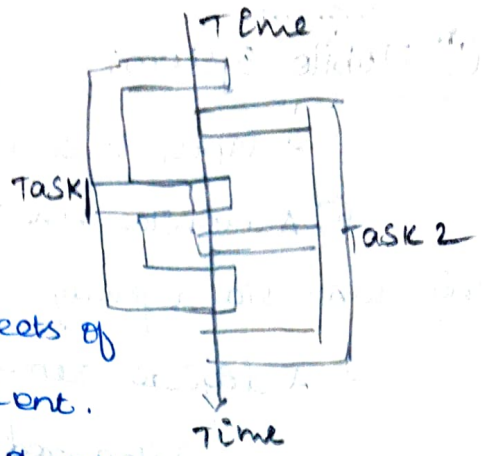
Operations that continue even in the presence of faults are referred to as fault-tolerant.

Techniques for dealing with failures:

- ① Detecting failures
- ② Masking failures
- ③ Tolerating failures
- ④ Recovering from failures
- ⑤ Redundancy.

6) Concurrency: - Shared resources at same time.

(*) Concurrent process.



7) Transparency:

Hide different aspects of distribution from the client.

It is the ultimate goal of many DS.

① Location transparency:

② Access Transparency:

③ Concurrency Transparency:

④ Replication Transparency

⑤ Failure transparency

⑥ Mobility Transparency

⑦ Performance Transparency

⑧ Scaling transparency:

Advantages: of transparency

① Easier for the user

② Easier for the programmer

Disadvantages of transparency

① Optimization can be done by programmer

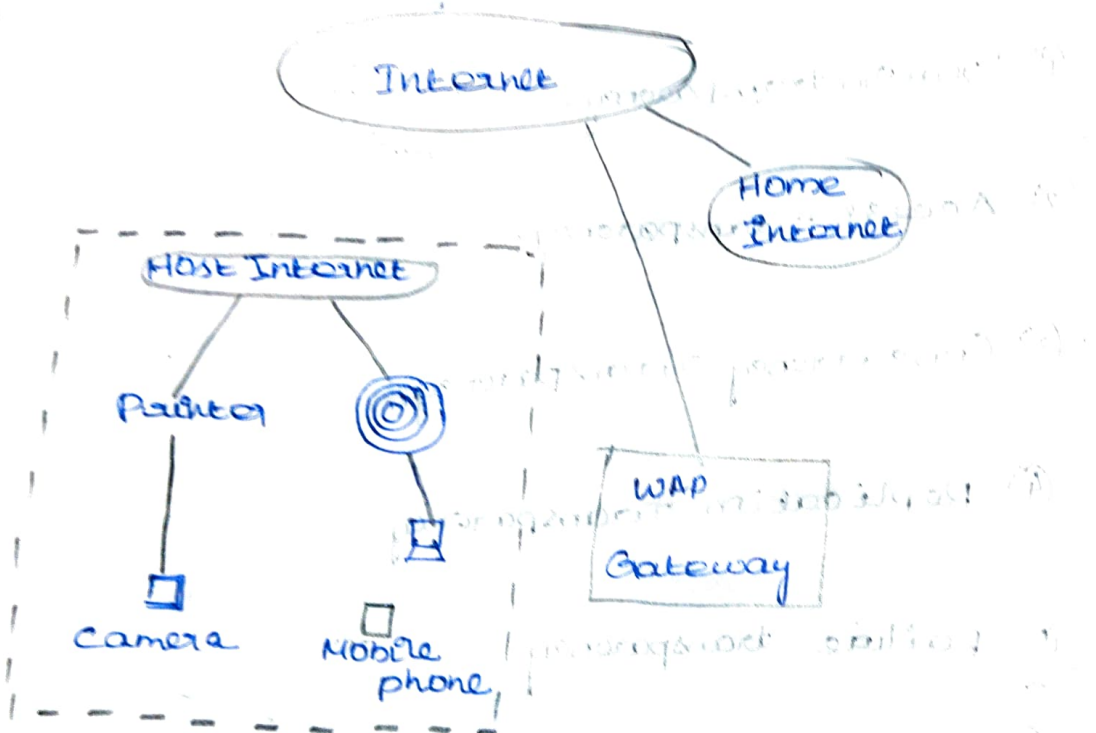
② Strange behaviour when the underlying s/m fails

③ Underlying s/m can be very complex

Application of Distributed Computing and Challenge

① Mobile System:

- ① A laptop is connected to host's wireless LAN
- ② A mobile phone is connected to Internet using WAP via a gateway.
- ③ A digital camera is connected to a printer over an Infra-red link.

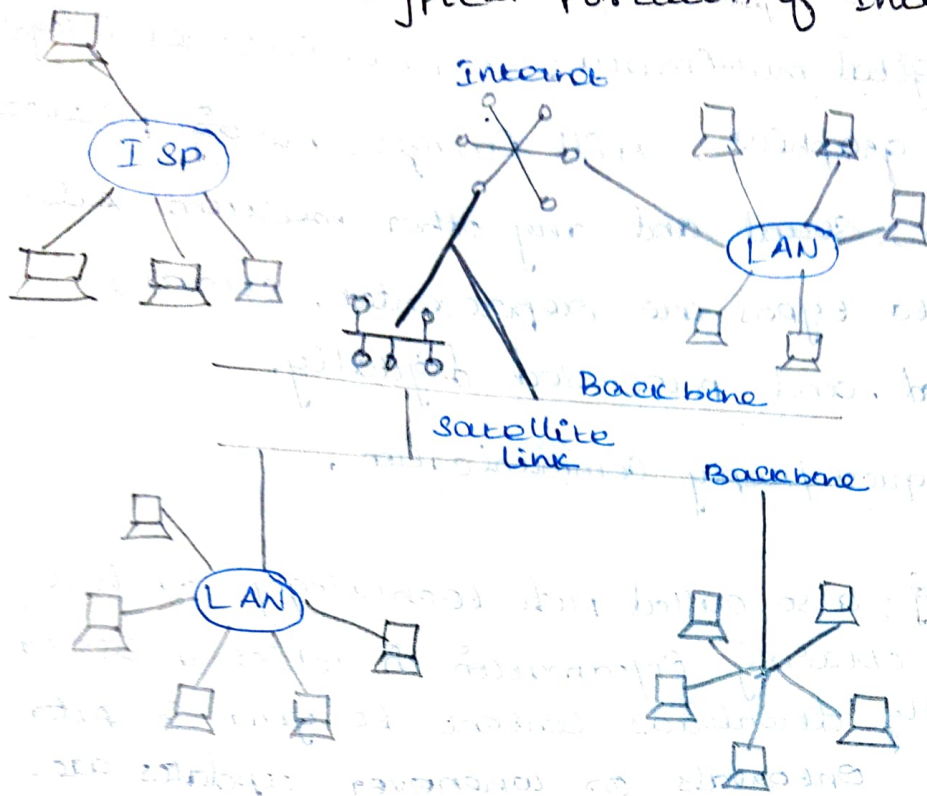


Portable devices in DS

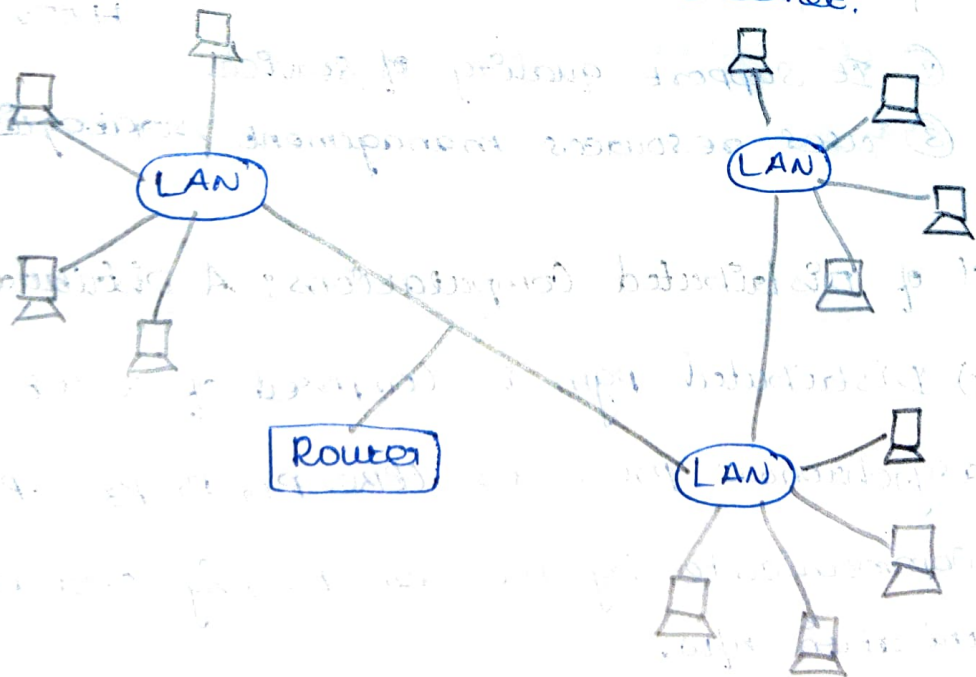
② Pervasive Computing:

- ↳ N/w has become a pervasive resource & devices can be connected at any time and any place.
- ↳ Prgms running on the computers, connected to Internet by passing message employing common means of communication.

Typical Portation of Internet.



Intranet: Intranet is composed of several local area n/w linked by backbone connections. Routers are used to connect Intranet to Internet.



Intranet

3) Multimedia System:

⇒ Digital multimedia: Computer Controlled Entanglement of text, graphics, still images, moving pictures, animation, sound and any other medium. All these data types are represented, stored, transmitted, and processed digitally.

⇒ Request/Reply Communication.

Web casting: also called push technology. It is a method of obtaining information in which a server automatically downloads content to your computer at regular intervals or whenever updates are made to the site.

Supports following Distributed Multimedia applications

① Encoding & encryption formats: eg. MP3, MPEG-1, MPEG-2, HDTV

② It supports quality of service

③ uses resource management strategies.

2) A Model of Distributed Computations: A Distributed P

(*) Distributed program is composed of a set of "n" asynchronous processes like $P_1, P_2, P_3, \dots, P_i, P_n$ that communicate by message passing over the communication n/w.

(*) Assume, each process is running on a different processor. The processes communicate only by using passing message method.

(*) Let c_{ij} , denote the channel from process p_i to process p_j and let m_{ij} denote a message sent by p_i to p_j . The comm. delay is finite & unpredictable.

(*) Processes do not share global clock.

(*) Process execution and msg transfer are asynchronous.

(*) Process sending msg is not wait for delivery of the msg to be complete.

A Model of Distributed Executions:

Process execution means sequential execution of its action. Actions are atomic.

Actions of process are three types.

① Internal Event: Affects only the process which is executing the event

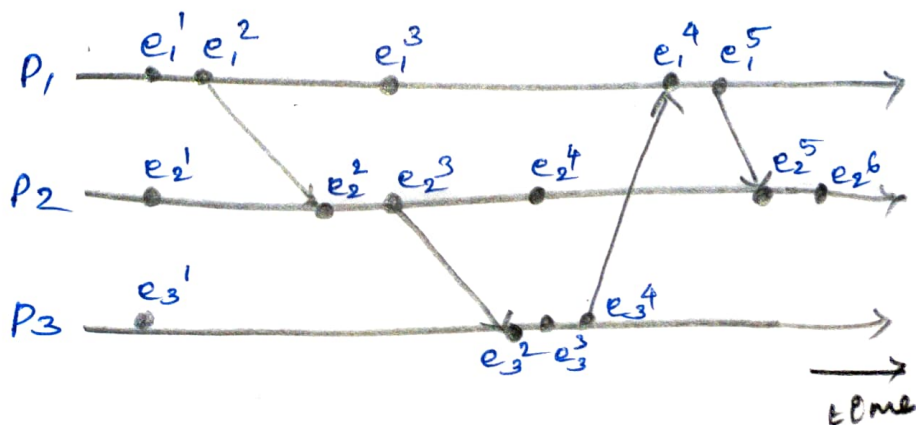
② Send Event: A process passes msg to other process

③ Receive Event: A process gets msg from other process

Message m ,

$\text{Send}(m)$ - Send Event

$\text{Rec}(m)$ - receive Event.



Causal Precedence relation:

Send & receive events signify the flow of info. b/w processes and establish causal precedence b/w events at the sender & receiver.

$$e_i \rightarrow e_j$$

Logical vs physical concurrency:

If two logical events are concurrent & only if they do not causally affect each other.

physical concurrency; ~~on the~~ the events at the same instant in physical time.

Model of Communication n/ws:

- ① FIFO - msg ordering
- ② Non-FIFO - random order.
- ③ Causal ordering

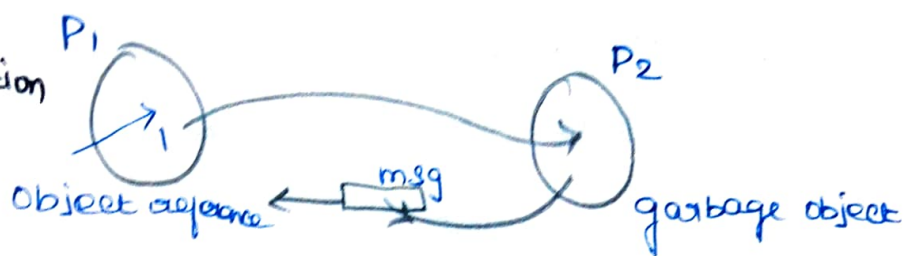
For any 2 messages m_{e_j} and m_{k_j} if $\text{send}(m_{e_j}) \rightarrow \text{send}(m_{k_j})$
then $\text{rec}(m_{e_j}) \rightarrow \text{rec}(m_{k_j})$

Global state of Distributed system:

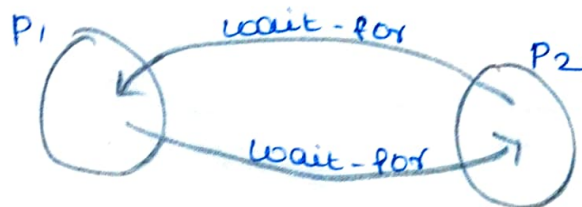
The global state of distributed computation is the set of local states of all individual processes involved in the computation plus the state of the comm. channel.

Requirements of global states:

a) Garbage collection



b) Deadlock



c) Termination



Unit - II

Logical time and Global state

①

- 1) Clock Events and process state
- 2) Logical Time
- 3) Physical clock synchronization: NTP
- 4) A Framework for a system of logical clocks
- 5) Scalar Time
- 6) Vector Time
- 7) Message ordering paradigms - M/D - 2022
- 8) Asynchronous Execution with synchronous communication
- 9) Synchronous pgm order on Asynchronous s/m
- 10) Group communication - Dec 2022
- 11) Causal order
- 12) Total order - Dec 2022
- 13) Global state & snapshot Recording Algm.
- 14) Snapshot Algms for FIFO channels - May 2022

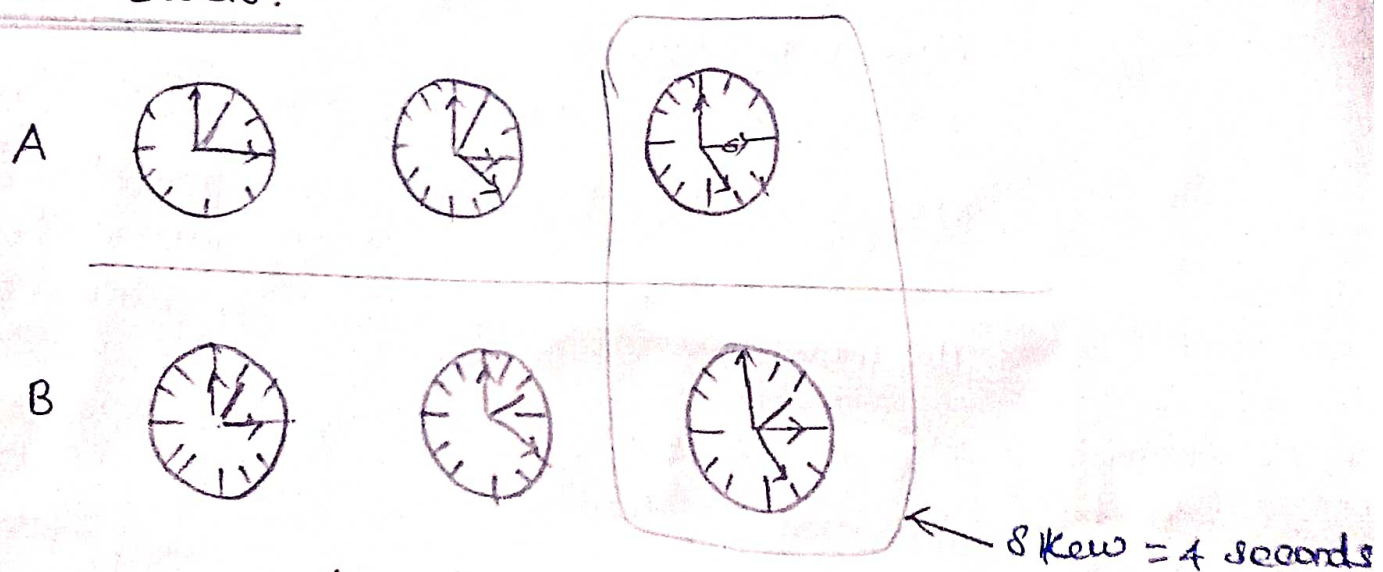
Clock events and process state:

- ↳ A DS is defined as a collection P of N processes P_i , $i = 1, 2, \dots, N$. Each process executes on a single processor and the processors do not share memory.
- ↳ Each process P_i has state S_i .
- ↳ Process communicate only by messages (via a n/w).
- ↳ Process P_i has to execute a seq. of actions.
 - ① Sending a msg
 - ② Receiving a msg
 - ③ Performing a computation that alters its state S_i

Send, Receive, change state. The seq. of events ^{working:} 1.0
2
single process P_i can be totally ordered, which is denoted by $e \rightarrow e'$ if and only if the event e occurs before e' at P_i .

History of process P_i to be the seq. of events that takes place within : $\text{history}(P_i) = h_i = \langle e_i^0, e_i^1, e_i^2, \dots \rangle$

Physical clock:

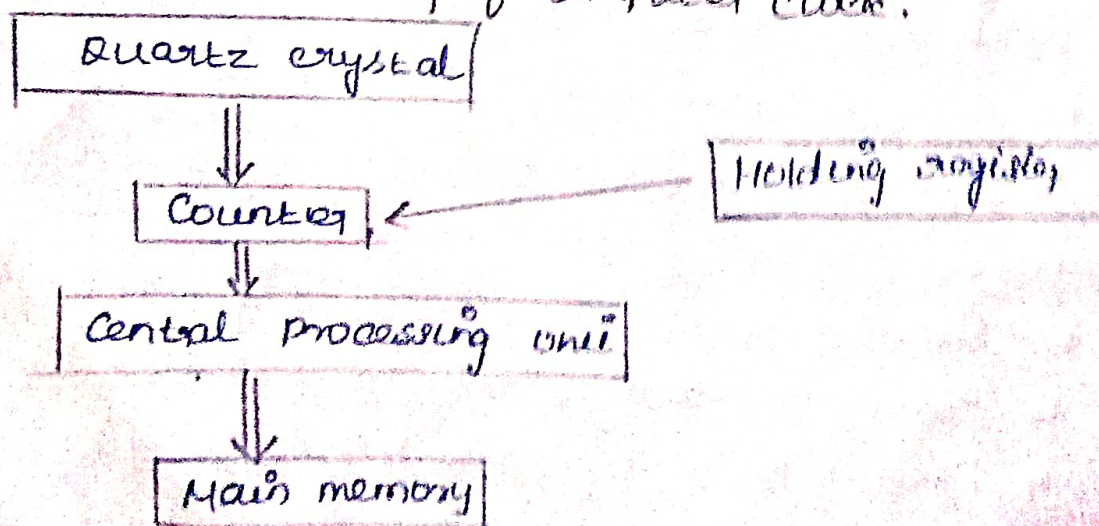


Clock drift and clock skew.

B is slightly faster (2 sec/per hour) than A.

4 seconds = 2 hours.

Working of Computer clock.



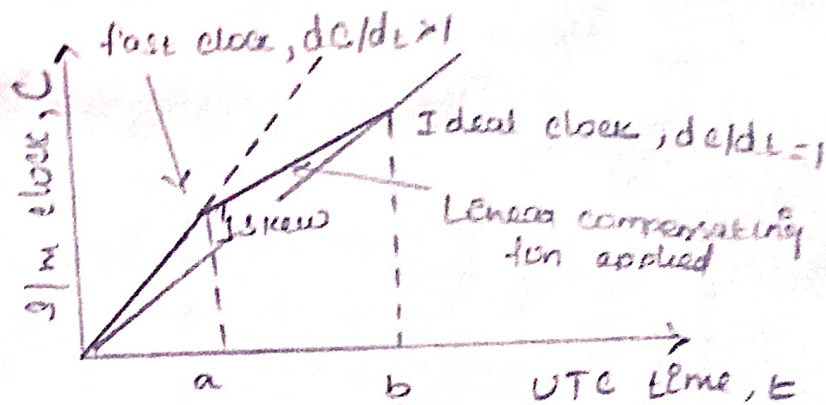
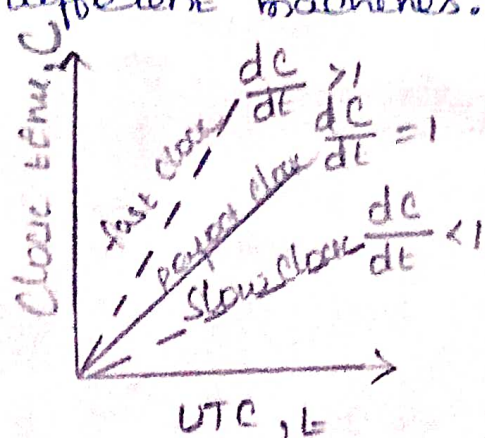
Timing:

(7)

1. Oscillation at a well defined frequency
2. Each crystal oscillation decrements the counter by 1
3. When counter gets 0, its value is loaded from the bidding register
4. When counter is 0, then an interrupt is generated, which is call clock tick.
5. At each clock tick, An interrupt service procedure add 1 to time stored in memory.

Clock skew and drift compensation:

The difference b/w two clocks at any point in time is called clock skew and is due to both clock drift and the possibility that the clocks may have been set differently on different machines.



Drift rate of clocks.

2) Logical Time:

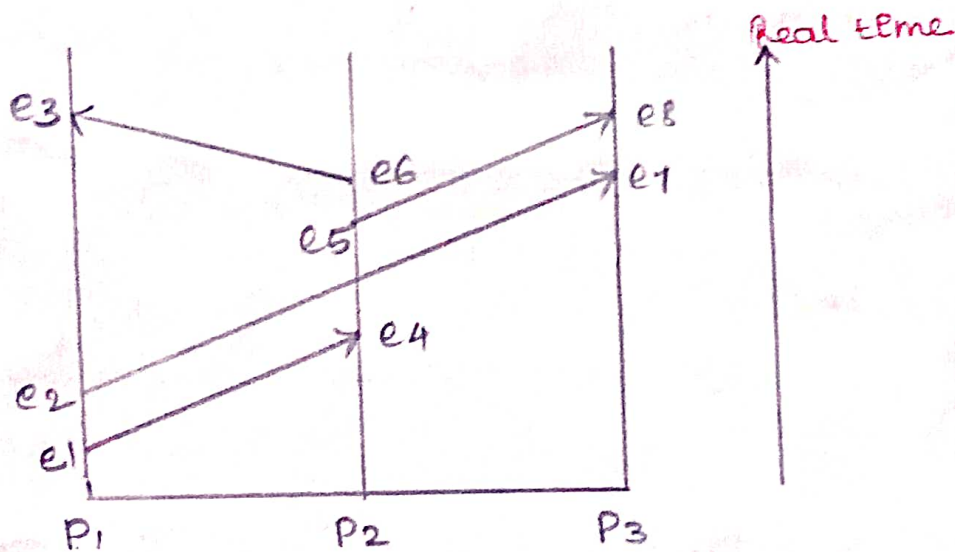
A logical clock C_p of process P is a slow counter that is used to timestamp events executed by P so that the happened-before relation is represented by the timestamps.

Event ordering:

Lamport defined a relation called happens before, represented by $\rightarrow$

Three Conditions:

- 1) If a and b are events in the same process, and a comes before b , then $a \rightarrow b$
- 2) If a is the sending event of a message msg by one process and b is the receipt event of msg, then $a \rightarrow b$.
- 3) If $a \rightarrow b$, $b \rightarrow c$, then $a \rightarrow c$



Event Ordering

Possible event ordering:

- 1) $e_1 \rightarrow e_2 \rightarrow e_3$
- 2) $e_1 \rightarrow e_4 \rightarrow e_5 \rightarrow e_6 \rightarrow e_3$
- 3) $e_2 \rightarrow e_7 \rightarrow e_8$

Event ordering is not possible in this condition:

$$e_2 \rightarrow e_4 \text{ and } e_5 \rightarrow e_7$$

Logical Clock Condition:

- 1) For any events a and b , if $a \rightarrow b$ then $C(a) < C(b)$
- 2) From the definition, the clock condition is satisfied if the following two conditions hold:
 - (i) Condition 1: If a and b are events of P_i , and a comes before b , then $C_i(a) < C_i(b)$
 - (ii) Condition 2: If a is the sending of a msg by P_i and b is the receipt of the msg by P_j , then $C_i(a) < C_j(b)$

2.2 Lamport Timestamp:

Lamport Algorithm is used to synchronize the logical clock. It uses happens-before relationship to provide a partial ordering events.

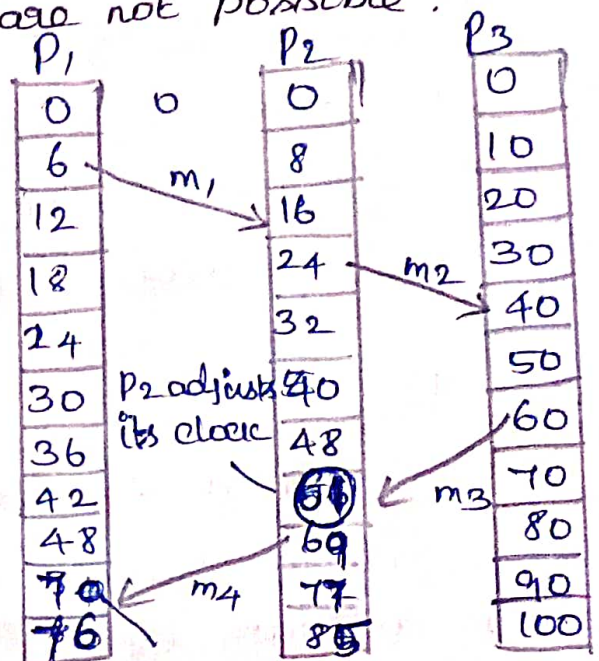
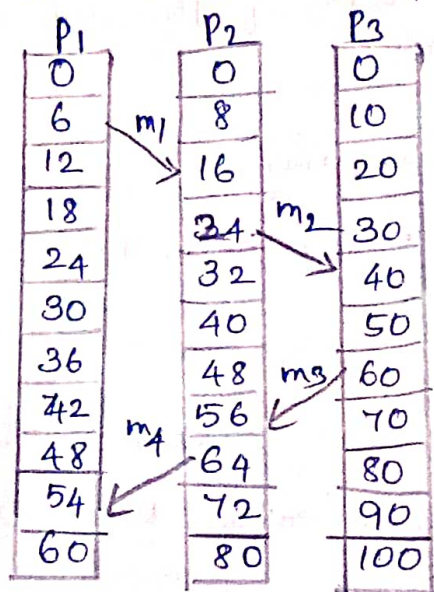
- ① All processes use a counter (clock) with initial value of zero
- ② The counter is incremented by one and assigned to each event, as its timestamp.
- ③ A send (msg) event carries its timestamp.
- ④ For receive (msg) event the counter is updated by
 - ⊗ $\text{Max}(\text{receiver-counter}, \text{message-timestamp}) + 1$

(*) When the clock has ticked 6 times in process P_1 ,
 " " " 8 " " " P_2 ,
 " " " 10 " " " P_3 .

- At time 6, process P_1 sends msg m_1 to process P_2 .
- The clock in process P_2 reads 10 when it arrives.
- Process P_2 will conclude that it took 10 ticks to reach from process 1 to process 2.

(*) According to this reasoning, msg m_2 from p_2 to p_1 takes 16 ticks.

Fig(a) Msg m_3 from p_3 to p_2 leaves at 60 and arrives at 56. Similarly, msg m_4 from p_2 to p_1 leaves at 64 and arrives at 54. The values are not possible.



a) Process with its own clock runs at different rate.

(b) Lamport's algorithm.

Fig(b) which uses happen-before relation. Since message m_3 left at 60, it must arrive at 61 or later. So each msg carry its sending time according to the sender's clock.

Limitations of Lamport's logical clock:

If $a < b$ then $c(a) < c(b)$, but if $c(a) < c(b)$ then $a \rightarrow b$ is not necessary true, (i.e) Concurrency information is lost.

④

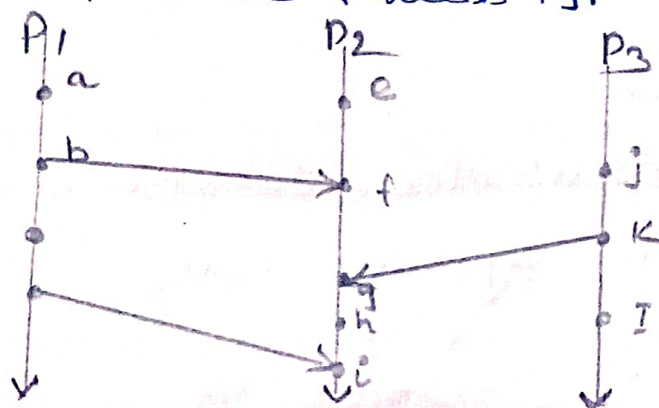
3 Vector Timestamp:

A vector clock vc is a mechanism that associates timestamps with events such that comparing two events timestamps indicates whether those events are causally related.

Represented as a vector $[c_1, c_2, \dots, c_n]$ with an integer clock value for each process.

Each process P_i has an array $VC_i[1..n]$, where $VC_i[j]$ denotes the no. of events that process P_i knows have taken place at process P_j .

Eg:

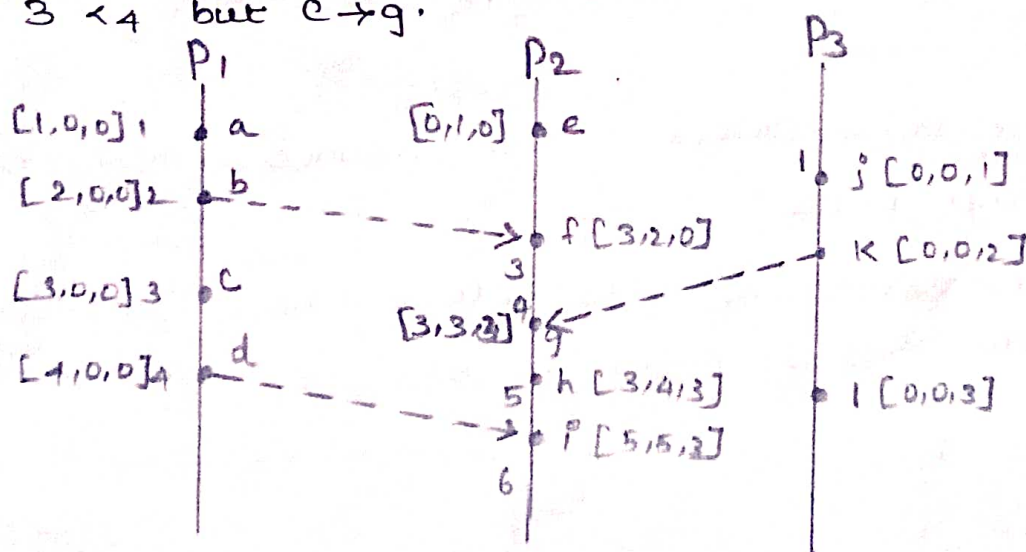


Assume that each process's logical clock is set to 0 initially

Solution: Lamport clocks:

$2 < 5$ since $b < h$;

$3 < 4$ but $e \rightarrow g$.



3) Physical clock synchronization: NTP

Clock synchronization is done by two methods

- ① Internal synchronization
- ② External synchronization

External synchronization:

External synchronization refers to syn. of process' clocks C_i with an authoritative external source S .

Let $D > 0$ be the syn. bound and S be the source of UTC. Then $|S(t) - C_i(t)| < D$ for $i = 1, 2, \dots, N$ and for all real times t .

Internal synchronization:

Refers to process' clocks C_i with each other.

Let $D > 0$ be the syn. bound and C_i and C_j are clocks at processes P_i and P_j , respectively.

Synchronization in a synchronous s/m:

Following bounds are defined:

- ① The time to execute each step of a process has known lower and upper bounds.
- ② Each msg transmitted over a channel is received within a known bounded time.
- ③ Each process has a local clock whose drift rate from real time has a known bound.

Cristian's Method for synchronizing clocks:

(5)

Cristian's Algorithm:

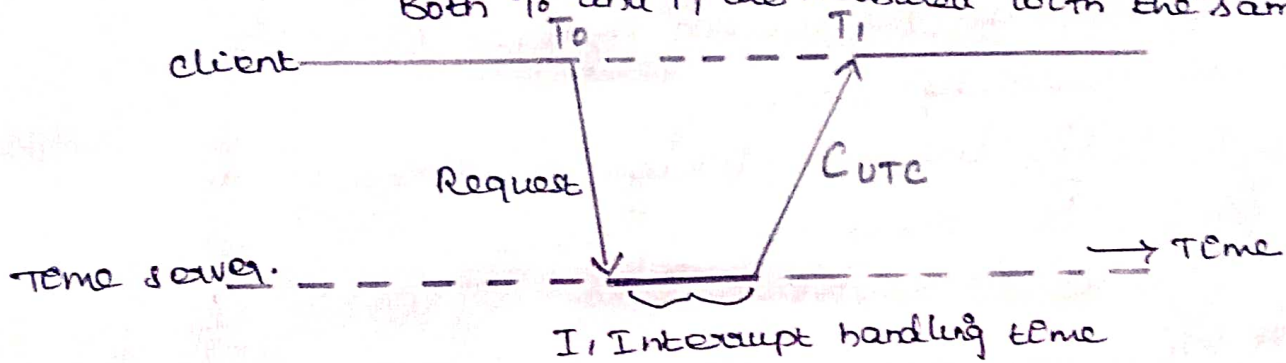
Cristian's Alg. is centralized passive time server type alg.

Algorithm ① A machine sends a req. to the server at least every $d/2s$ seconds, where d is the max^m difference allowed b/w a clock and the UTC.

② The time server sends a reply msg.

③ Machine measures time delay b/w server sending msg time and machine's receiving the msg.

Both T_0 and T_1 are measured with the same clock



message propagation time = $(T_0 + T_1)/2$

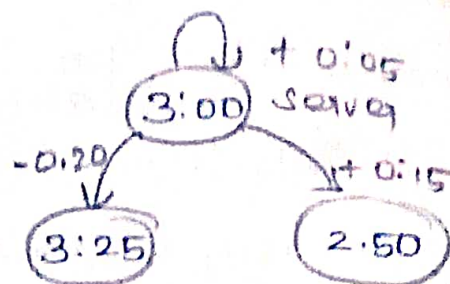
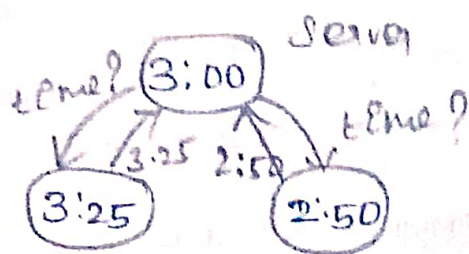
new time can be set:

$$T_{\text{new}} = T_{\text{server}} + (T_0 + T_1)/2$$

Berkely Algorithm:

⇒ Time server is active machine.

⇒ The server polls each machine periodically, asking it for the time. The time at each machine may be estimated by using Cristian's method to account for n/w delays.



Network Time Protocol (NTP)

NTP is yet another method for synchronizing clocks that uses a hierarchical architecture where the top level of the hierarchy are servers connected.

Goals:

- Reliable service
- Provide protection against interference.
- Increasing Accuracy.

- ① Multicast on high speed local LANs
- ② Procedure call mode
- ③ Symmetric mode.

NTP servers synchronize modes are:

1) Multicast mode:

- ⇒ Intended for use on a high-speed LAN
- ⇒ One or more servers multicasts the time to servers running in other computers connected by the LAN.

2) Procedure-call mode:

- ⇒ One server accepts requests from the other computers, which it processes by replying with its time.

3) Symmetric mode:

- ⇒ Master servers that supply time info. in LANs.
- ⇒ Achieves highest accuracy.

1) A Framework for a system of Logical clocks: ⑥

A S/m of logical clocks consists of a time domain T and logical clock C . Elements of T form a partially ordered set over a relation $<$.

Relation $<$ is called the happened before or causal precedence.

$$C: H \rightarrow T$$

Implementing logical clocks:

It requires addressing following issues.

- ① Data structures local to every process to represent logical time.
- ② Protocol to update the DS to ensures the consistency condition.

Each Process P_i maintains two capabilities:

- a) Local logical clock helps process P_i measures its own time.
- b) Logical global time.

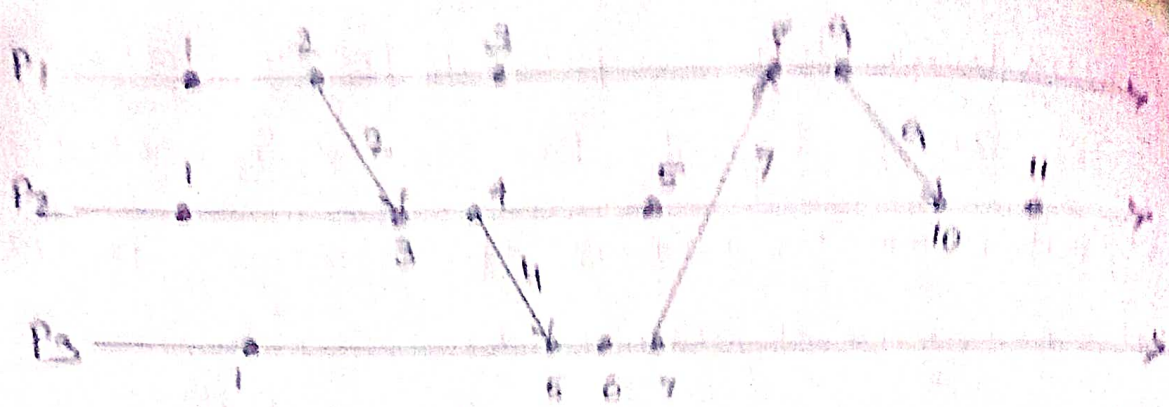
Logical time has many applications.

- a) used to resolve conflicts.
- b) Debugging distributed S/ms.

5) Scalar Time:

⇒ It is designed by Lamport to synchronize all the events in DS.

⇒ The logical local clock of a process P_i and its local view of the global time are squashed into one integer variable C_i .



Then space-time diagram

Properties:

- ① Consistency property: For two events e_i and e_j
 $e_i \rightarrow e_j \Rightarrow c(e_i) < c(e_j)$
- ② Total ordering:
- ③ Event Counting:
- ④ System of scalar clocks is not strongly consistent.

6) Vector Time:

Determine whether pairs of events are causally related.

A vector clock of a s/m N processes is an array/vector of N logical clocks, one clock per process.

- ① Initially all clocks are zero
- ② Each process experiences an internal event, it increments its own logical clock in the vector by one.
- ③ Each time for a process to send a msg, it must increment its own clock
- ④ Each time for a process receives a msg, it increments its own logical clock in the vector by one and update the element.

Message Ordering Paradigms: - AP/May - 2022

⑥

N/D - 2022

⇒ Order matters is important

⇒ Msg orders are

(*) FIFO

(*) Causal order,

(*) Synchronous order.

① Asynchronous execution is an execution for which the causality relation is a partial order.

② FIFO execution is an A-execution in which for all (s, r) and $(s', r') \in T$ $(s \sim s' \wedge r \sim r' \text{ and } s < s') \Rightarrow r < r'$

All physical links obey FIFO rule.

③ Non-FIFO - Connection-Oriented Service. eg: TCP

④ Causally Order execution is an A-execution in which for all (s, r) and $(s', r') \in T$, $(r \sim r' \text{ and } s < s') \Rightarrow r < r'$

8) Asynchronous Execution with Synchronous Communication:

Synchronous order: All the comm. b/w pairs of processes is by using synchronous send and receive primitives.

(*) Comm. program for an asynchronous s/m deadlocks when using syn. primitives.

Process A

.....

Send (A)

Receive (B)

.....

Process B

.....

Send (A)

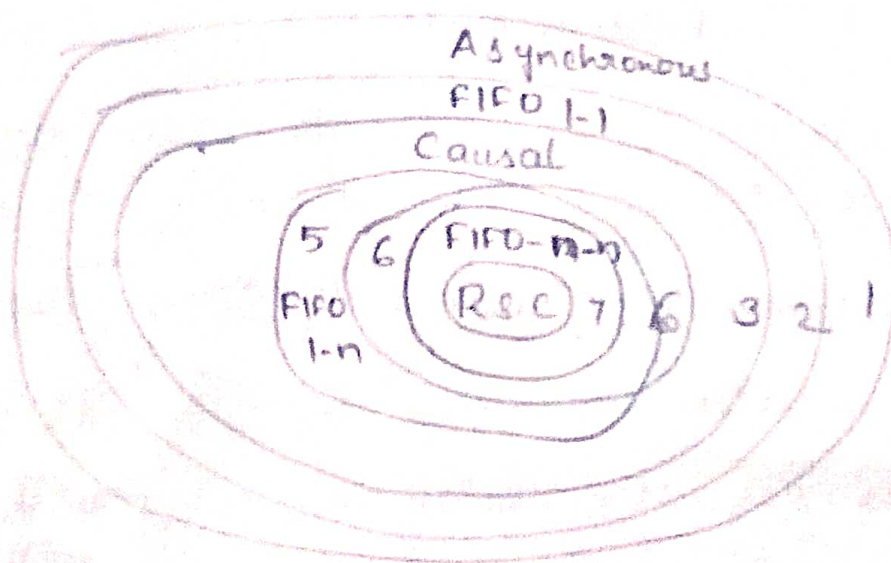
Receive (A)

.....

Execution Realizable with Synchronous Comm:

An exe. is realizable with syn comm. if send event is immediately followed by its corresponding receive event.

Hierarchy of message ordering paradigms:



→ The executions which are valid for a given model may also be valid for another model.

→ An A-execution is realizable with synchronous comm, if and only if A is an S-execution.

→ A pgm using syn. comm. easiest to develop and verify. A pgm using non-FIFO comm, resulting in an A-execution, hardest to design and verify.

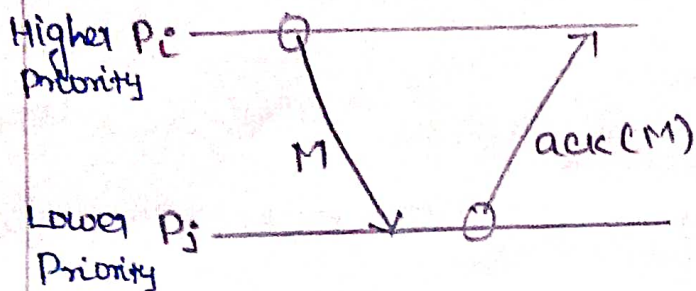
Synchronous Pgm order on Asynchronous s/m: (9)

Bagrodia's Alg for Binary rendezvous:

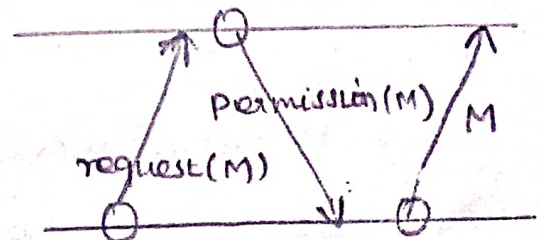
Assumptions:

- ① Receivers are always enabled
- ② Send, once enabled, remains enabled.
- ③ To break deadlock, PIDs used to introduce asymmetry.
- ④ Each process schedules one send at a time.

Msg types: M , $ack(M)$, $request(M)$, $permission(M)$



a) High Priority process block



b) Low priority process does not block.

10) Group Communication - NOV/DEC - 2022

It can be implemented in several ways.

- ① One to many
- ② Many to one
- ③ Many-to many.

One to Many Communication:

- (*) Msg is sent by one sender to multiple receivers.
- (*) It also known as multicast comm.

Group Mgmt: - Receiver processes forms a group. Two types are closed group - Only the members of group can send a msg

open group - Any process in the s/m can send a msg to the group as whole.

Buffered and unbuffered multicast:

↓
Msg is not buffered
for receiving process

↓
Msg is buffered for the awaiting
process.

Flexible reliability in multicast comm:

- a) 0-reliability:
- b) 1-reliability
- c) m-out-of-n-reliable:
- d) All reliable:

2) Many-to-One Communication:

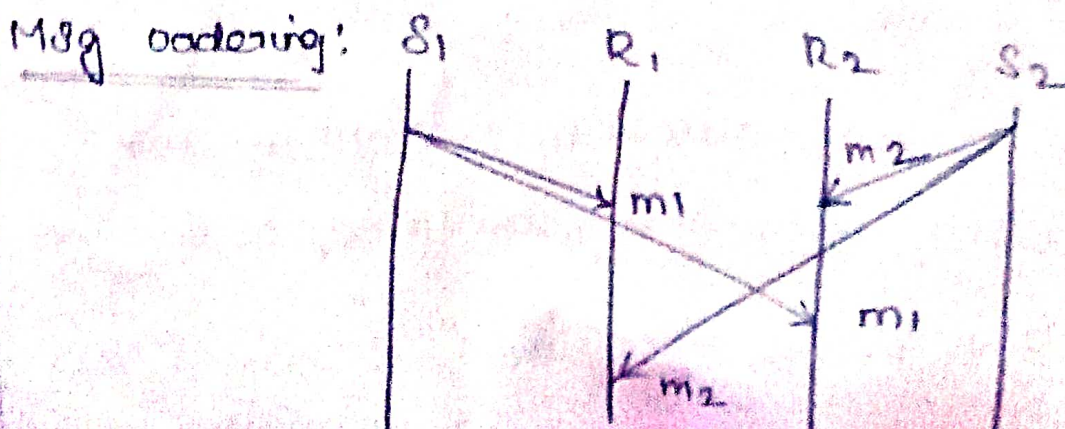
Multiple senders send msg to a single receiver.
Single receiver may be selective or nonselective.

- ① Selective receiver: Specifies a unique sender and msg exchange takes place only if that sender sends a msg.
- ② Non-selective receiver: Specifies a set of senders & if any one sender in the set sends a msg to this receiver a msg exchange takes place.

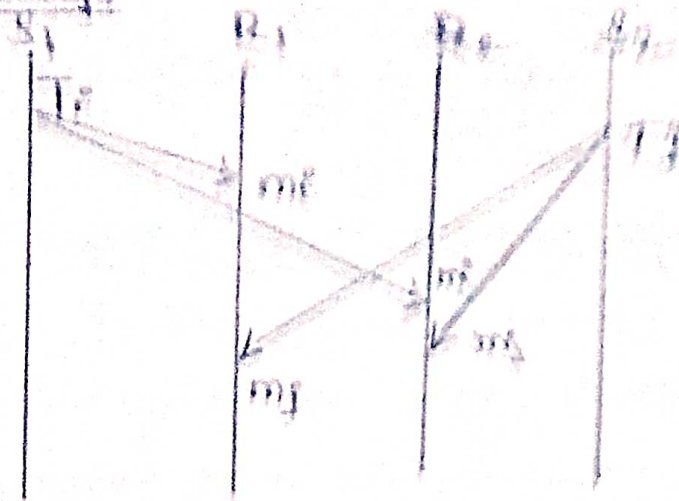
3) Many-to-many Communication:

⇒ Multiple sender and multiple receivers.

Msg ordering:

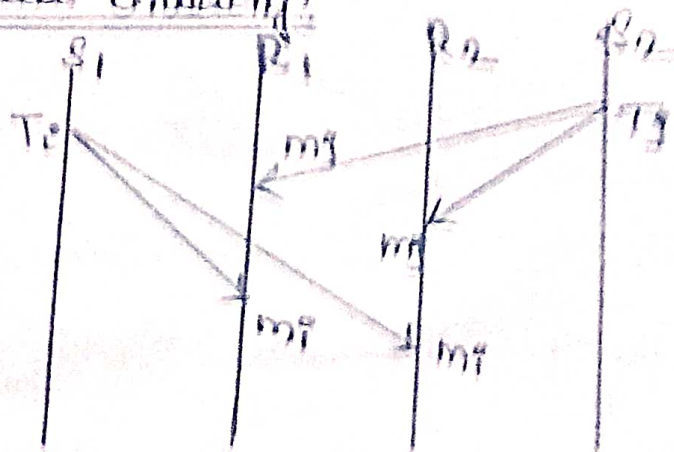


① Absolute Ordering:



In this ordering, all msgs are delivered in the same order as they were sent. Processes in the system order in which they were sent.

Consistent/Total Ordering:



Msg received in the same order, regardless of their timestamp.

Causal Ordering:

(*) If two msg sending events are not causally related, the two msgs may be delivered to the processes in any order.

(*) Two msg sending events are said to be causally related if they are correlated by the happened before relation.

Drawbacks: ① Very as an overhead,

② Broadcast assumed.

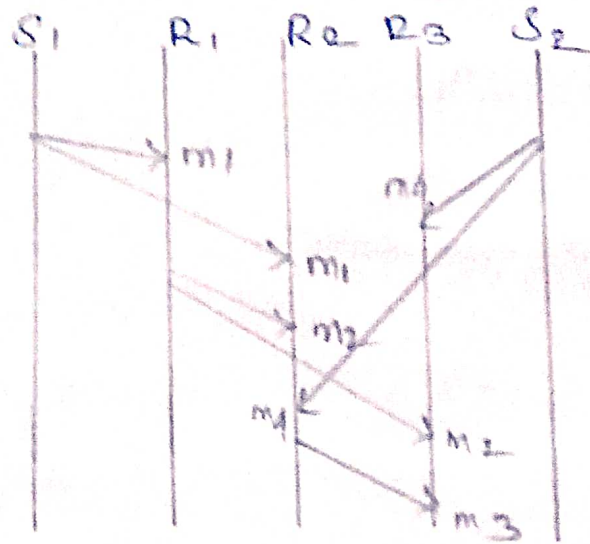
n) Causal Order:

⇒ Causal ordering of msgs was proposed by Birman and Joseph.

⇒ Multiple sender send messages to multiple receiver.

Ordered msg delivery ensures that all msg are delivered to all receivers in an order acceptable to the appli.

⇒ The purpose of causal ordering of msgs is to ensure that the same causal relationship for the "msg send" events correspond with "msg received" events.



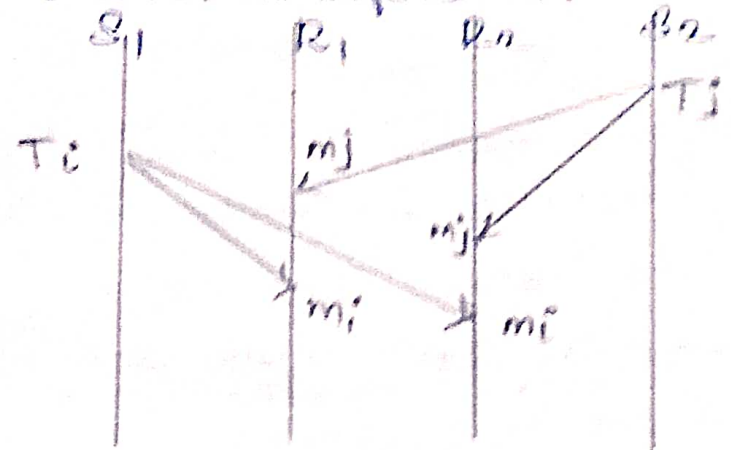
Raynal-Schiper Towses Algorithm:

(*) In this alg, each process p_i maintains an $n \times n$ matrix, $SENT_i$, where n is the total no. of processes to record the no. of msgs, as i has enm, sent from each process to each other's.

Total order For group

It ensures that all msg are delivered to all receiver processes in the same order.

If a correct process delivers msg "m" before it delivers "m'", then any other correct process that delivers m' will deliver m before m'.



Rule: : msg received in the same order, regardless of their timestamp.

Implement: A msg sent to a sequencer, assigned a sequence number is finally multicast to receivers.

Example: Replicated database updates.

Drawback: A Centralized Alg.

Three phase Distributed Algorithm:

Sender:

Phase 1: Msg m with locally unique tag is local broadcast to the group members.

Phase 2: The sender collects a reply from all group members.

Phase 3: The process multicast the final timestamp to the group in line.

Receiver:

Phase 1: Receiver receives the msg with a tentative timestamp.

Phase 2: The receiver receives sends the received TS back to sender.

(*) Final timestamp is received from multiple

13) Global State and Snapshot Recording Algorithms:

(*) An asynchronous DS consists of several processes without common msg which communicate solely via msg with unpredictable transmission delays.

(*) Global State and Global Time is an important role.

(*) An Event Structure is a pair $(E; <)$, where E is set of events, and $<$ is an irreflexive partial order on E called the causality relation.

System model:

$\Rightarrow$ There is no physical clock in the s/m. msg send and receive is asynchronous.

$\Rightarrow$ Let C_{ij} denote the channel from process i to process j .

$\Rightarrow$ The state of channel C_{ij} , denoted by SC_{ij} , is given by the set of msgs in transit in the channel.

1) Snapshot Algorithms for FIFO Channels: - A/M-2022 (10)

1) Chandy - Lamport Algorithm: - A/M/2022

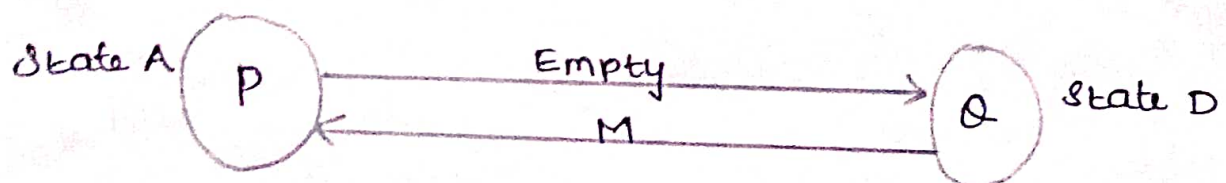
This algorithm records a set of process and channel state such that the combination is a consistent global state. Comm. channels assumed to be FIFO.

Algorithm:

- 1) Initiator process P_0 records its state locally.
- 2) Marker sending rule for process P_i :
- 3) Marker receiving rule for process P_i :

if (P_i has not yet recorded its state) it
else (P_i has already recorded its state)

Property of the recorded Global State:



⇒ Assume that the state P is recorded in global state S_0 , so the state recorded for P is A . After recording its state, P sends a marker along channel C .

⇒ On receiving the marker, Q records its state, which is D , and records the state of C to be the empty seq.

Global state S^* recorded by the alg is not identical to any of global states S_0, S_1, S_2, S_3 .

Unit - II

Distributed Mutex and Deadlock

1) Distributed Mutual Exclusion Algorithms: Introduction

Mutual Exclusion: Makes sure that concurrent process access shared resources or data in a serialized way. If the process P_i is executing in its critical section, then no other processes can be executing in their critical sections.

Entry section: The code executed in preparation for entering the critical section

Critical section: The code to be protected from concurrent access

Exit section: The code executed upon leaving the critical section

Remainder section: The rest of the code.

2) Preliminaries:

System Model:

↳ The s/m consists of N sites, $S_1, S_2, \dots, S_N$.
We assume that a single process is running on each site.
The process at site S_i is denoted by P_i .

↳ Site may in one of the three states:

(*) Requesting CS

(*) Executing CS

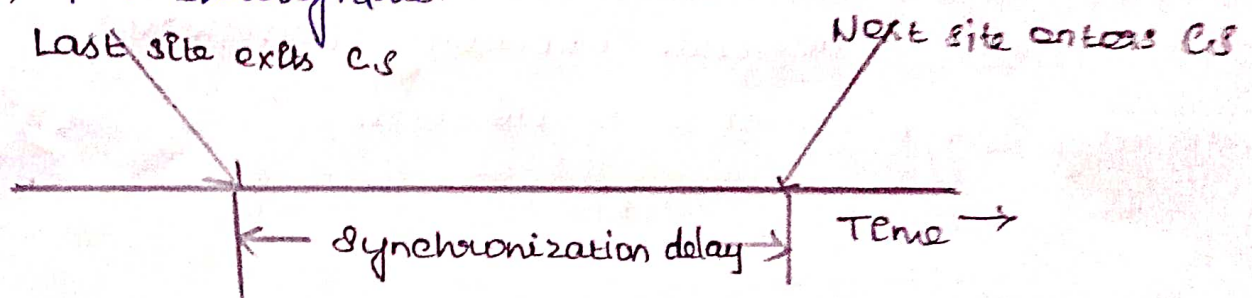
(*) Neither requesting nor executing req. for CS

Requirements for Mutual Exclusion:

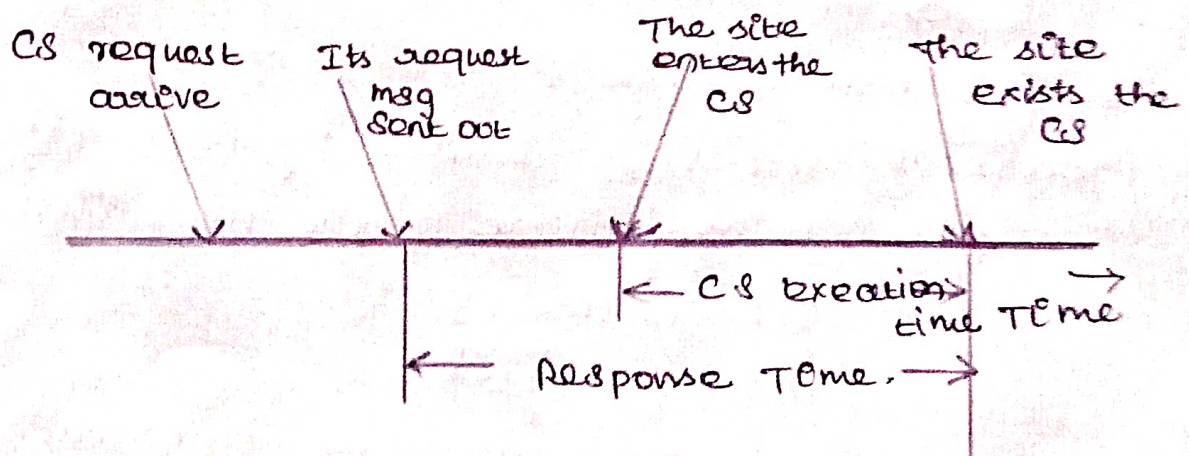
- 1) Freedom from deadlocks
- 2) Freedom from starvation
- 3) Strict fairness
- 4) Fault tolerance.

Performance Metrics:

- 1) Message Complexity,
- 2) Synchronization delay,
- 3) Response time
- 4) S/m Throughput.



Synchronization delay



Response Time.

5) Lamport's Algorithm:

Each process fairly and equally competes for the right to use the shared resource requesting are arbitrated by central control site or by distributed agreement.

Requesting the critical section:

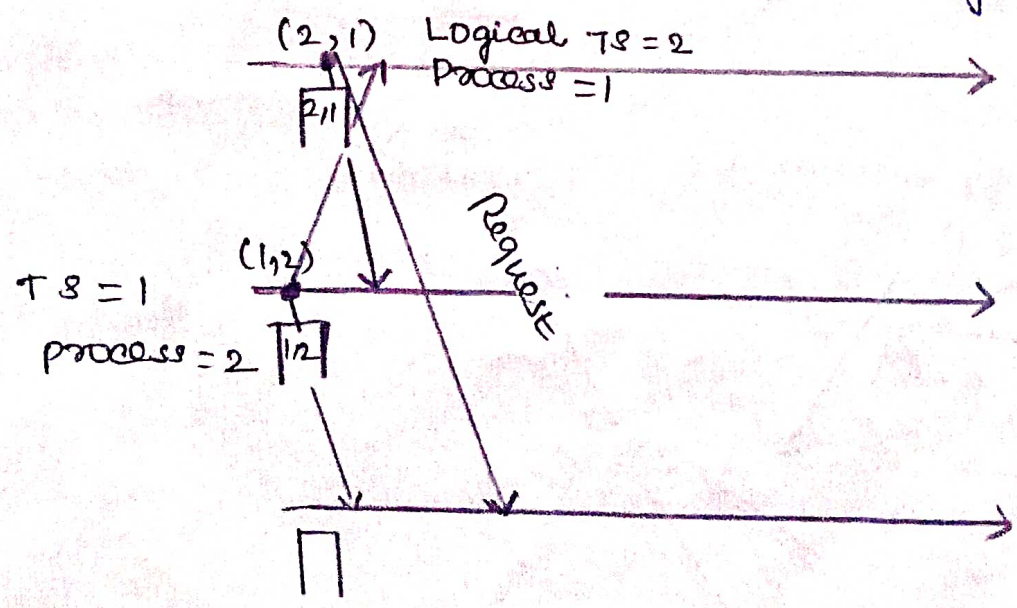
- ⇒ Send a REQUEST(C_i, i) to each other site, placed in a
- ⇒ When S_i receives REQUEST(C_i, i) it returns a timestamped REPLY to S_i and places the REQUEST in request queue.

Conditions for entering CS:

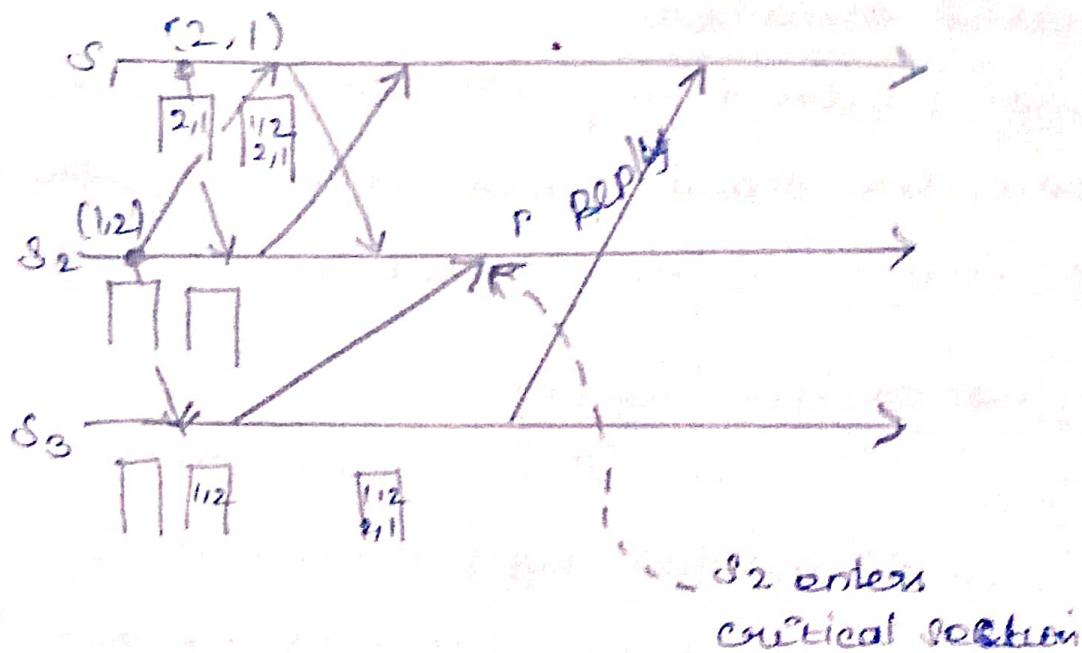
- L1: S_i has received a msg with timestamp larger than (C_i, i) from all other sites.
- L2: S_i 's request is at the top of the request queue

Optimization:

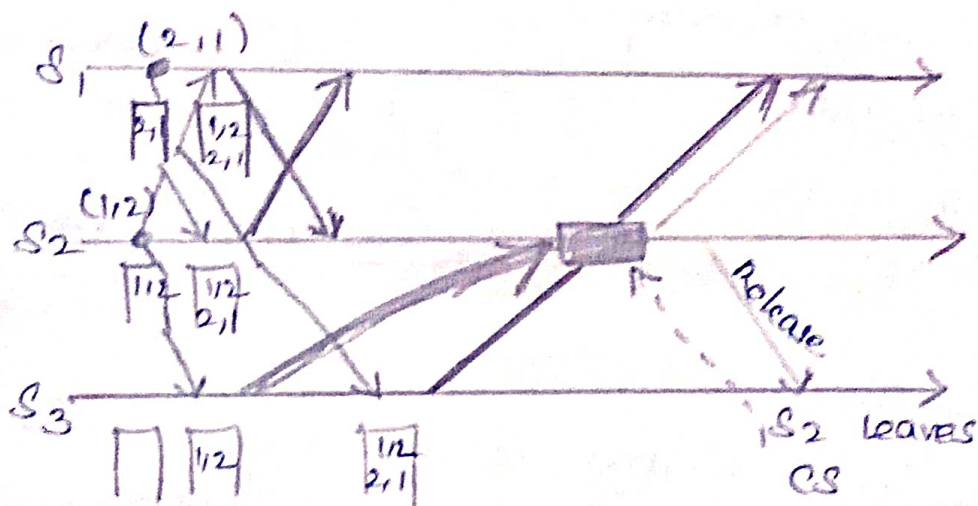
Step 1: Sites S_1 and S_2 are making req. for critical section



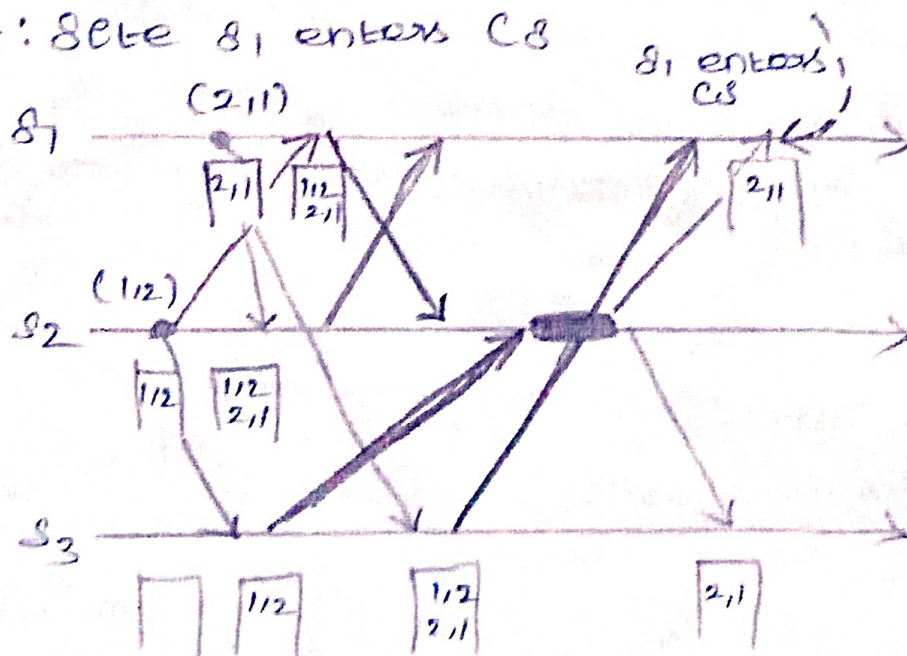
Step 2: Site S_2 enters the critical section.



Step 3: Site S_2 exits CS and sends RELEASE msg.



Step 4: Site S_1 enters CS



(3)

Ricore - Agrawala's Alg:

A/M-2022

N/O-2022

It is an optimization on Lamport Alg.

It uses only two types of msgs:

① REQUEST

② REPLY

It is assumed that all processes keep a logical clock which updated according to the clock rules.

Algorithm:

Requesting the CS:

① Req when S_i wants to enter the CS, it broadcasts TS REQUEST msg to all site.

② When a process receives REQUEST msg. It may enter the states.

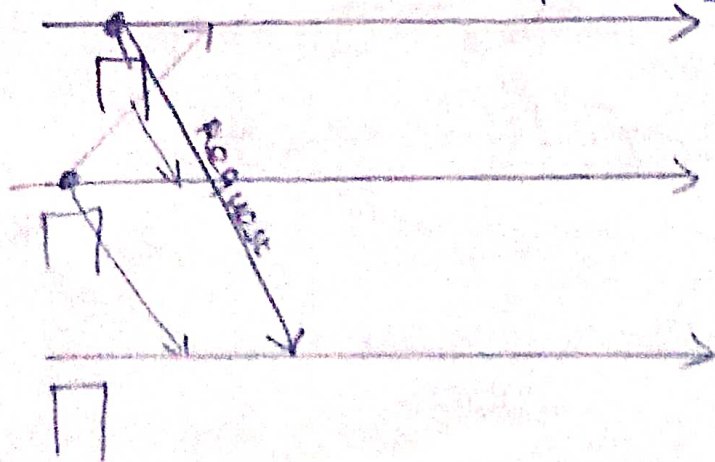
Executing the CS:

When the site S_i enters CS after it has received REPLY msg from the all site in its request set.

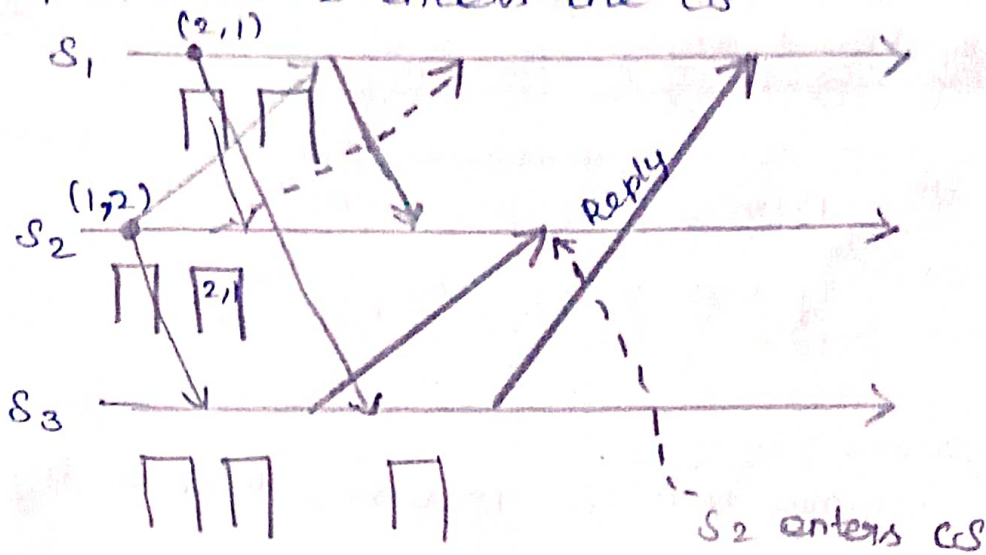
Releasing the CS:

When site S_i exits the CS, it sends RELEASE(C_i) msg to all sites in its request set R_i .

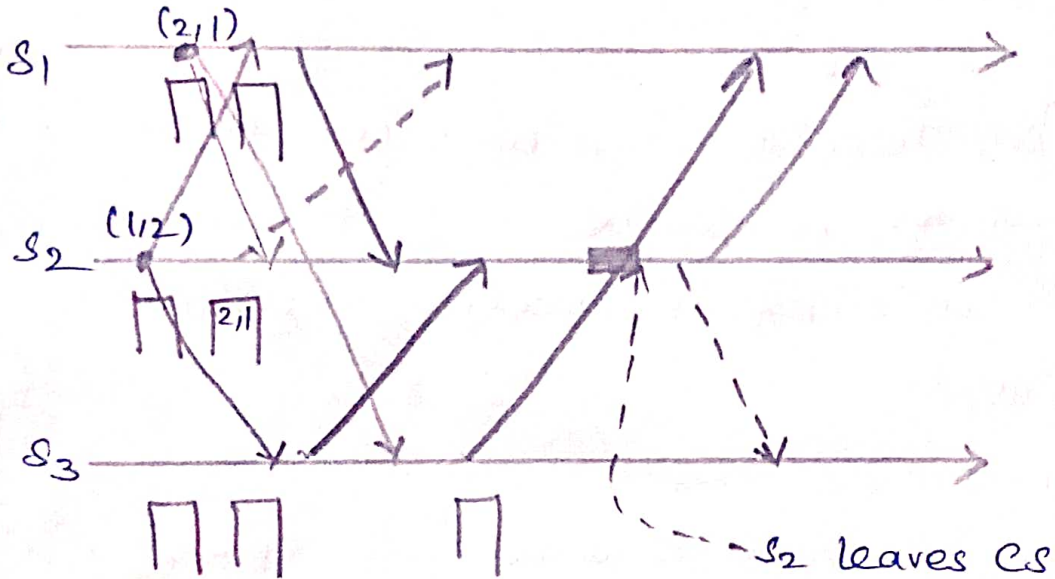
Optimization: Step 1: Site S_1 and S_2 are making req. for CS



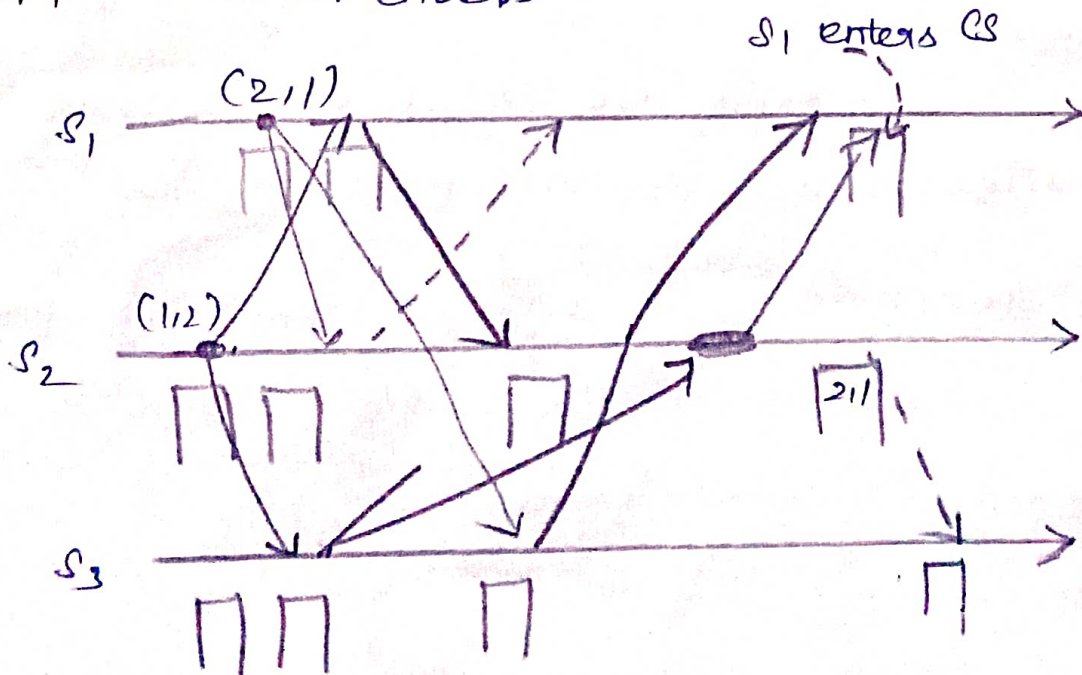
Step 2: Site S_2 enters the CS.



Step 3: Site S_2 exits the CS and sends a REPLY msg to S_1 .



Step 4: Site S_1 enters to CS



Token based Algorithms: - AV N/D - 2022

⇒ A unique token is shared among all the sites in DS.

⇒ Two Algorithms.

1) Suzuki-Kasami's Broadcast Alg:

⇒ A unique token is shared among all the sites. This means a seq. no. is used instead of a TS.

⇒ To enter CS, a site broadcasts its REQUEST msg to all other sites.

Major Design Issues:

⇒ NO RELEASE/REPLY msg as in assertion based protocols

⇒ The site with the token needs to know which site to send the token next.

Algorithm:

Requesting CS:

1) Site S_i increments its seq. no. $RN[i]$ and sends a REQUEST (i, S_n) to all other sites with seq. no. S_n .

2) When site S_j receives this msg, it sets $RN[j] = \max(RN[j], S_n)$.

Executing CS:

3) Site S_i executes CS when it has received the token

Releasing CS:

4) $LN[i] \leftarrow RN[i][i]$.

5) For each S_j not in the token queue, append i to queue if $RN[i][j] = LN[j] + 1$

6) If token queue is non empty after update, delete top ID and send to site by ID

5) Deadlock Detection in DS: Introduction - A/M/2

A DS consists of a no. of s/m connected by a system. Each s/m maintains some of resources of the s/m.

If the resource is local, the process makes a request of the local controller.

However, each site's WFG could be cycle free and yet the DS could be deadlocked. This is called global deadlock situation.

① Process A at site 1 holds a lock on resource X.

② Process A ~~at site~~ has requested, but has not been granted, resource Y at site 2.

③ Process B at site 2 holds a lock on resource Y.

④ Process B has requested, but has not been granted, resource X at site 1.

Necessary condition:

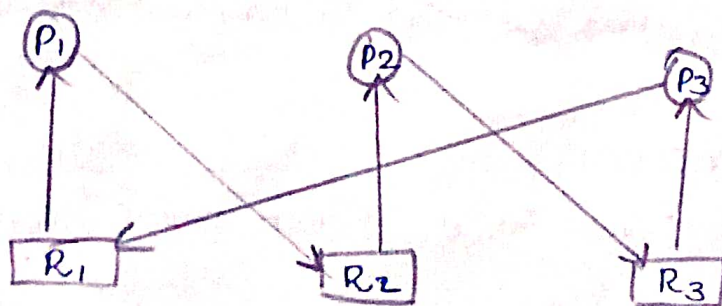
Two states: Running or blocked.

① Mutual exclusion - Only one process may use resource at a time.

② No pre-emption - Hold resource and req. another one.

③ Hold and wait - waits for another process to release resource.

④ Circular wait - No resource can be holding.



These deadlocked processes.

2) System Model:

Resource type $R_1, R_2, \dots, R_m$

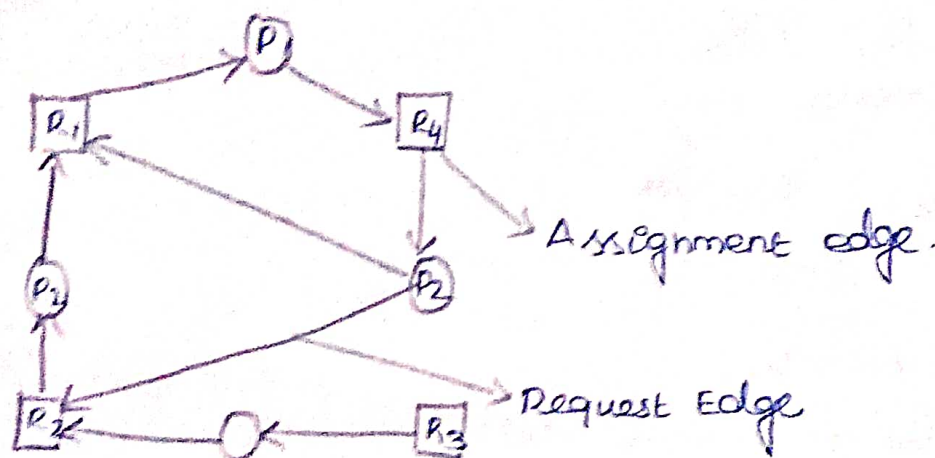
Each resource type R_i has w_i instances.

Process utilizes a resource as follows.

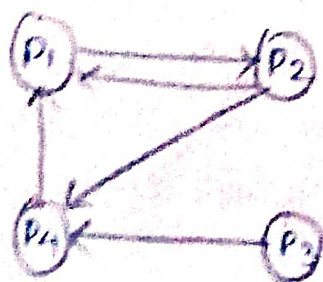
- ① Request ② Use ③ Release.

Wait for Graph:

A s/m is deadlocked if its resource allocation graph contains a directed cycle or a kna.



In DS, the s/m state can be modeled or represented by a directed graph, called "wait-for graph". Edges from node P_1 to node P_2 if P_1 is blocked and is waiting for P_2 to release some resources.



A s/m is deadlocked if and only if there is directed cycle or not in the WFG.

Resulting wait for graph of resource allocation graph is shown

8) Preliminaries: Deadlock handling strategies.

Three deadlock handling strategies

① Deadlock Prevention:

(*) First method:

→ prevent the circular wait condition by defining a linear ordering of resource types.

Disadv: resources cannot be requested on the orders that are needed.

(*) Second method:

prevent the hold-and-wait condition by requiring the process to acquire all needed resources before starting execution.

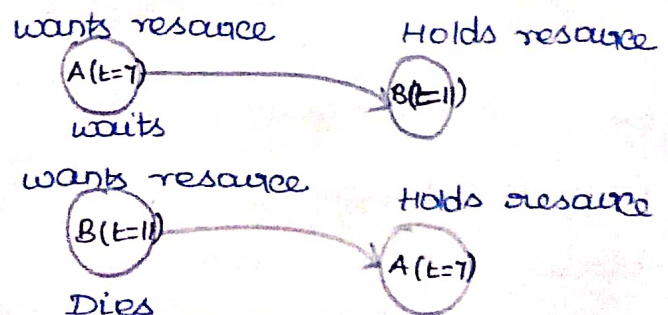
Disadv: Inefficient use of resources
Reduced concurrency

(*) Third method:

Use of time-stamps:

1) Wait-die method: A non-preemptive approach.

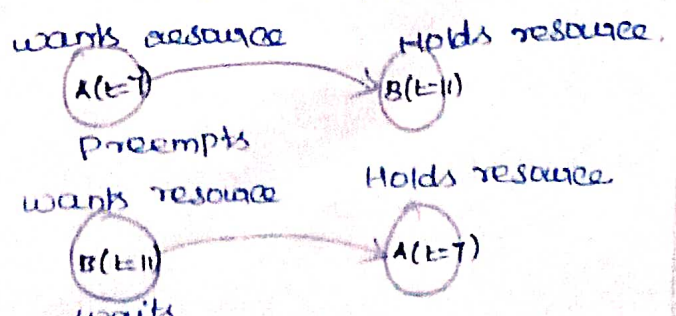
```
if  $[e(T_2) < e(T_1)]$   
  halt- $T_2$  ('wait');  
else  
  kill- $T_2$  ('die');
```



wait-die method.

2) Wound-~~wait~~ method: It is pre-emptive approach.

```
if  $[e(T_2) < e(T_1)]$   
  kill- $T_1$  ('wound');  
else  
  halt- $T_2$  ('wait');
```



Dead Avoidance:

Decision made dynamically, before allocating a resource, the resulting global s/m state is checked, if it is safe state then allow for allocation.

Disadvantages:

- ① Every site has to maintain global state of s/m.
- ② Different sites may determine that state is safe, but global state may be unsafe.

Deadlock Detection:

Principle of operation: Detection of a cycle in WFG proceeds concurrently with normal operation.

- ① The alg must detect all existing deadlock in finite time
- ② The alg should not report non-existent deadlock

Models of Deadlocks:

Models of deadlocks distributed s/m allows different kind of resource requests so; that means, they are represented by different model might require a single resource.

The single resource model:

A process can have at most one outstanding request for only one unit of a resource.

The AND model

→ Set of deadlocked processes, where each process waits for a resource held by another process.

→ Uses AND condition.

AND Condition: A process that requires resources for execution can proceed when it has acquired all resources.

AND condition for deadlock in s/m using the AND condition is the existence of a 'cycle'

⇒ A process can have at most one outstanding request.

The OR model:

⇒ Set of deadlocked processes, where each process was to receive messages from other processes in the set.

⇒ Uses OR condition.

OR condition: A process that requires resources for execution can proceed when it has acquired at least one of them.

The AND-OR-model:

⇒ A generalization of the previous two models is AND-OR model.

⇒ In the AND-OR-model, request may specify any combination of and and or in resource request.

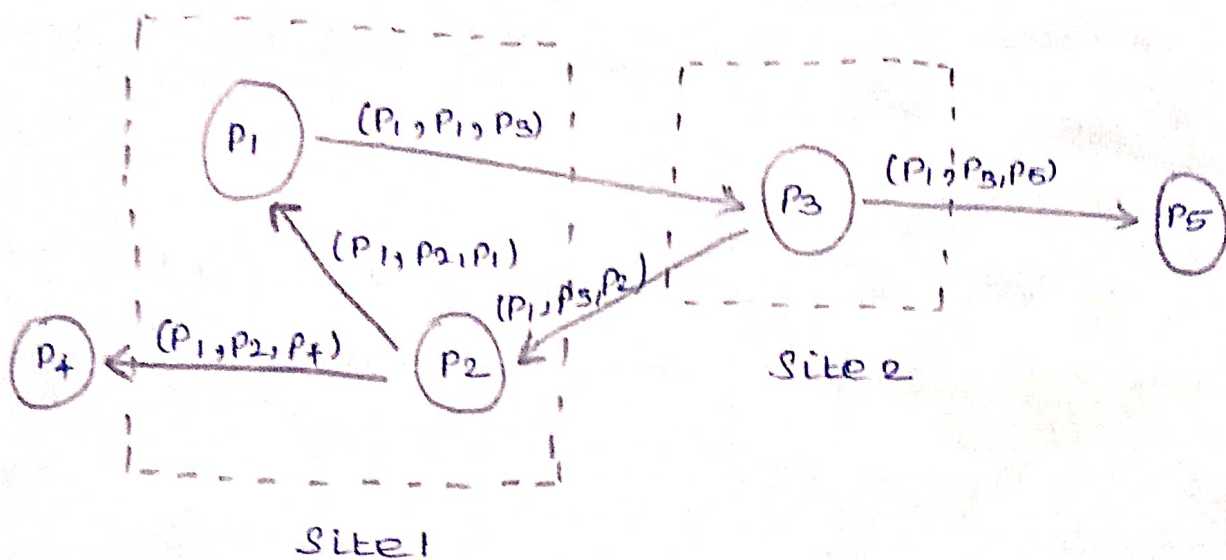
Handy - Misra - Hass Algorithm for the AND model:

⇒ This is considered an edge-chasing, probe based algorithm. It is also considered one of the best deadlock detection algorithms for distributed s/ms.

⇒ The process makes a request for a resource. With fails or times out, the process generates a probe msg.

Each probe msg contains the following info:

- ① The id of the process that is blocked.
- ② The id of the process is sending this particular version of the probe msg.
- ③ The id. of the process that should receive this probe msg.



Advantages:

- It is easy to implement.
- Each probe msg is of fixed length.
- There is very little computation.
- There is very little overhead.
- There is no need to construct graph.
- This alg does not find false deadlock.
- There is no need for special data structures.

10) Chandy - Mista - Haas Algorithm for OR Model!

It is distributed deadlock detection alg. for OR model is based on the approach of diffusion-computation. A blocked process determines if it is deadlocked by initiating a diffusion computation.

⇒ Two types of msgs are used in a diffusion computation:

(*) Query(i, j, k)

(*) Reply(i, j, k) denoting belongs to a diffusion computation initiated by process p_i and being sent from process p_j to process p_k .

⇒ A blocked process initiates deadlock detection by sending query msg to all process in its depends set. If an active process receives a query or reply msg, it discards it.

Actions:

1) If this is the first query msg received by p_k for the deadlock detection initiated by p_i , then it propagates the query to all the processes in its dependent set a local variable.

2) If this is not the engaging query, then p_k returns a reply msg to it immediately provided p_k has been continuously blocked since it received the process corresponding engaging query.

Unit - IV

Consensus and Recovery

Consensus and Agreement Algorithms: Problem Definition

Processes/sites in distributed systems often complete as well as cooperate to achieve a common goal.

In distributed data bases, there may be a situation where data managers have to decide "whether to commit or abort the Transaction".

Presence of failure: processors can fail or misbehave intentionally. Several rounds of message exchanges might be needed before agreement can be reached.

Byzantine Agreement Problem:

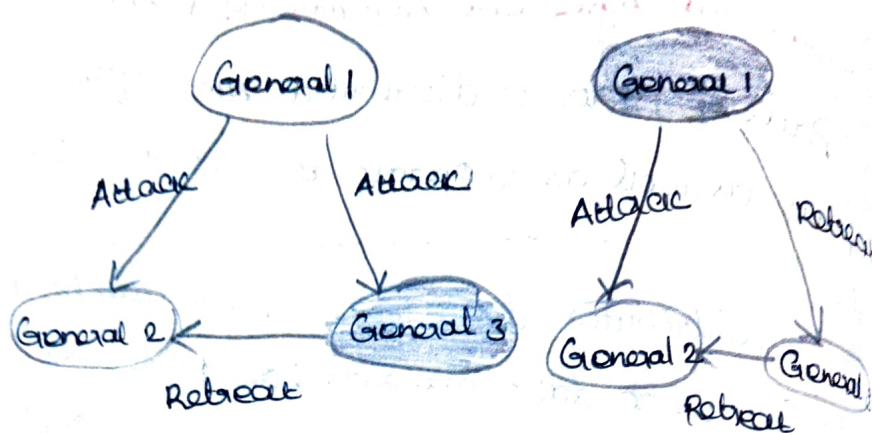
Several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. After observing the enemy, they must decide upon a common plan of action.

Three or more generals are agree to attack or to retreat. Once the commander is issue the order, lieutenants to the commander are to decide to attack or retreat.

But the one or more of the generals may be treacherous, i.e. faulty.

Agreement: All non-faulty processors agree on the same value.

Validity: If source is non-faulty, then the common agreed value must be the value supplied by source processor.



Consensus Problem:

(*) Every processor broadcasts its initial value to all other processors.

(*) Initial values may be different for different processors.

(*) Every processor has its own initial value.

(*) All non-faulty processors must agree on a single common value.

1) Agreement: All non-faulty processors agree on the same single value.

2) Validity: If the initial value of every non-faulty processor is V , then the common agreed value of all non-faulty processors must be V .

Interactive Consistency Problem:

- 1) Every processor has its own initial value.
- 2) All non faulty processors must agree on a set of common values.

Overview of Results:

S. NO	Failure mode	Synchronous s/m	Asynchronous s/m.
1.	No failure	Agreement attainable	Agreement attainable
2.	Crash failure	Agreement attainable [$f < n$ process]	Agreement not attainable.
3.	Byzantine failure	Agreement attainable [$f \leq (n-1)/3$ byzantine process]	Agreement not attainable

Solution to Byzantine Agreement problem:

- (*) First defined and solved by Lamport.
- (*) An arbitrary source processor broadcasts its initial value to all others.
- (*) If the source processor is faulty, other non-faulty processor can agree on any common value.
- (*) Faulty processors values and agreements do not matter.
- (*) Number of faulty processors, m , cannot exceed: $\text{trunc}[(n-1)/3]$

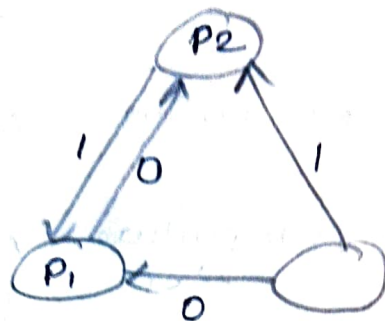
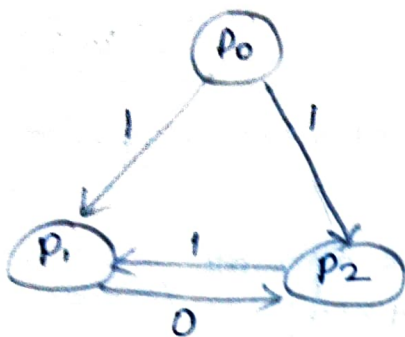
Impossible Scenario:-

Consider a s/m with 3 processors: P_0

Two possibilities:

Case 1: P_0 (Source) is not faulty, P_2 is faulty.
 P_1 should agree upon 1 as the value.

Case 2: P_0 is faulty, P_1 may agree on 1 and P_2



Lamport - Shostak - Pease Algorithm:

⇒ This algorithm also known as "Oral Message Algorithm $OM(m)$ " where m is the number of faulty processors.

⇒ n = Number of processors and $n \geq 3m+1$

⇒ Algorithm $OM(0)$

1) Source processor sends its value to every processor.

2) Each processor uses the value it receives

from source.

⇒ Algorithm $OM(m)$, $m > 0$

1) Source x broadcasts value to all processors.

2) Let v_i = value received by process i from source. process i acts as a new source and initiates $OM(m-1)$, sending v_i to remaining $(n-2)$ processes.

Time Complexity = $m+1$ rounds

Message complexity = $O(n^m)$

Lamport's Algorithm:

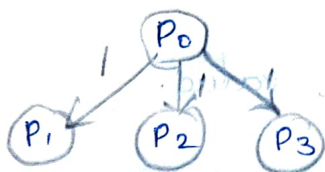
S/m with 4 processors: P_0, P_1, P_2, P_3 . P_0 is source
 P_2 is fault

Assumption: possible values are only 1 and 0.

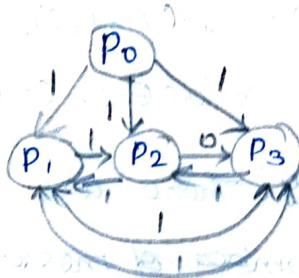
Step 1: P_0 initiates the initial value to be 1

Step 2: $OM(0)$. P_1 sends 1 to $\{P_2, P_3\}$. P_3 sends 1 to $\{P_1, P_2\}$

Step 3: Majority function at P_1 and P_3 is 1, which is the desired result.



Processor P_0 executes the alg. $OM(1)$



Processor P_1, P_2, P_3 executes the alg. $OM(0)$

Agreement in a failure-free s/m (Synchronous and Asynchronous).

In a failure s/m, consensus can be reached by collecting info from the different processes, arriving at a "decision" and distributing this decision in the s/m.

Synchronous s/m: This is implemented in a constant number of rounds. Common knowledge of the decision value can be obtained using an additional round.

Asynchronous s/m: Consensus can similarly be reached in a constant number of message hops.

Agreement in Synchronous slm with failures

Algorithm for Consensus with up to f processes in a slm of n processes are as follows:

Int f ;

Integer; $x \leftarrow \text{local value}$;

Process p_i executes the consensus alg.

for round from 1 to $f+1$ do

if the current value of x has not been broadcast(x);

$y_j \leftarrow \text{value}$

$x \leftarrow \min \forall j (x, y_j)$.

Output x as the Consensus value.

Complexity: There are $f+1$ rounds, where

The number of messages is at most $O(n)$ each round, and each message has one

Hence the total number of messages is

$$O((f+1) \cdot n^2)$$

Introduction of Check pointing and Rollback

Checking pointing and rollback are well-known techniques that are processes make progress in spite of failures. The failures under consideration are transient problems such as hardware errors and transaction ~~errors~~ aborts.

(*) When failure occurs, the process rolls back to its most recent checkpoint, restores the state saved in that checkpoint and resumes execution.

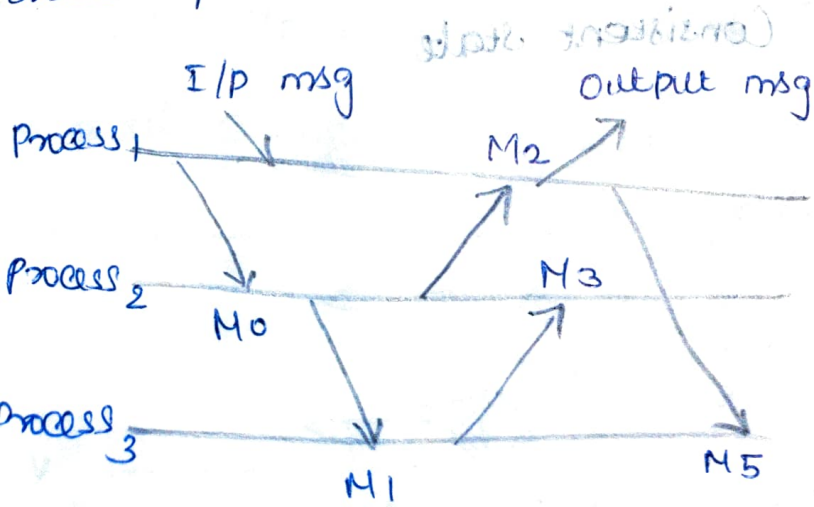
(*) Rollback recovery treats a ds. application as a collection of processes that communicate over a n/w. The saved state is called a checkpoint and the procedure of restarting from a previously checkpointed state is called rollback recovery.

Background and Definitions:

System model:

We consider the ds with fixed no. of processes ($P_1, P_2, \dots, P_n$). They are communicating with each other by using messages. The processes do not share a common msg or a common block.

(*) Rollback recovery protocol generally makes assumptions about the reliability of the inter-process communication.



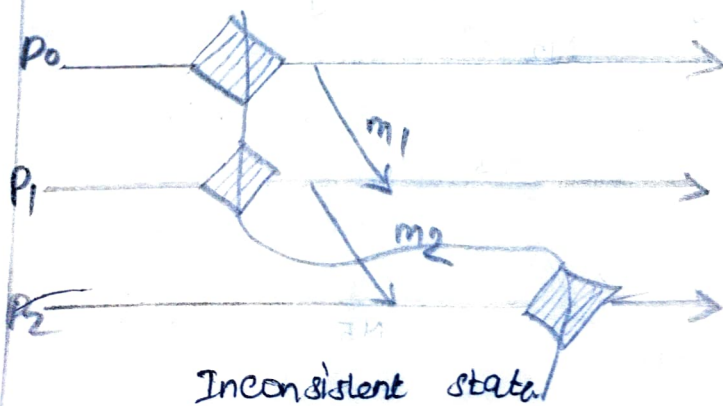
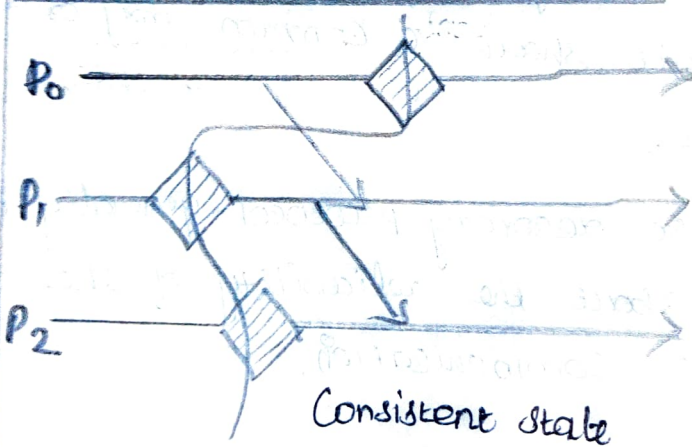
The Computation is asynchronous (i.e.) each process progresses at its own speed and msgs are exchanged through reliable channels, whose transmission delays are finite but unpredictable.

Local checkpoint:

Local checkpoint is a snapshot of the state of the process at a given instance and the act of recording the state of a process is called local checkpointing.

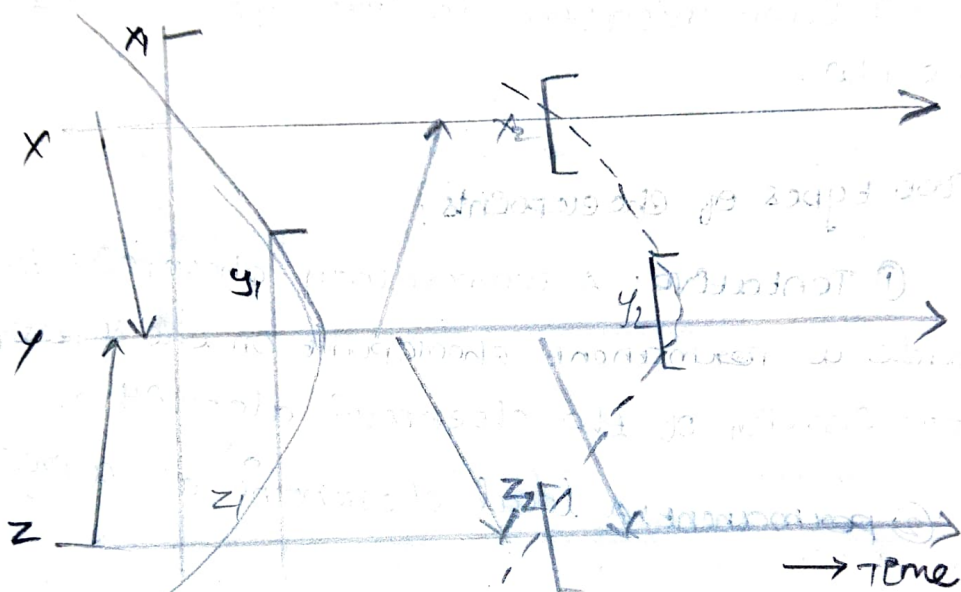
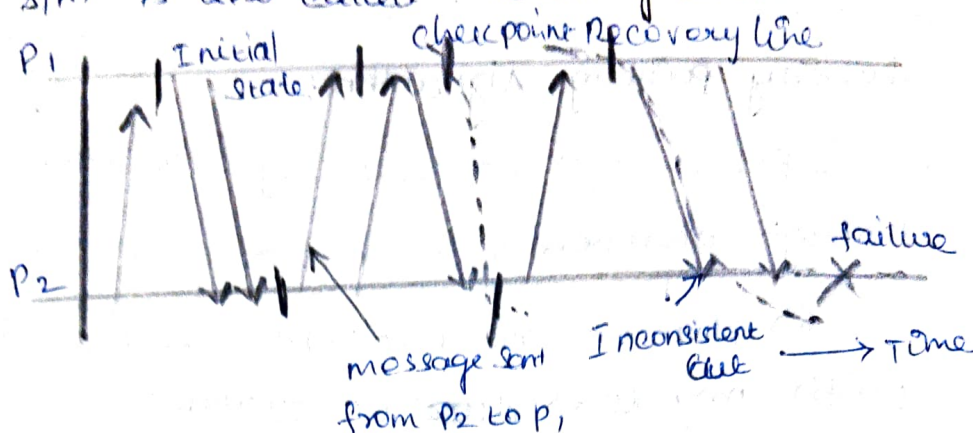
Each checkpoint is taken by a process and is assigned a unique seq. number. The i th ($i \geq 0$) checkpoint of process P_i is assigned a seq. no. i and is denoted by $C_{p,i}$.

Consistent set of checkpoints:



Strongly consistent set of checkpoints:

The most recent distributed snapshot in a s/m is also called recovery line.

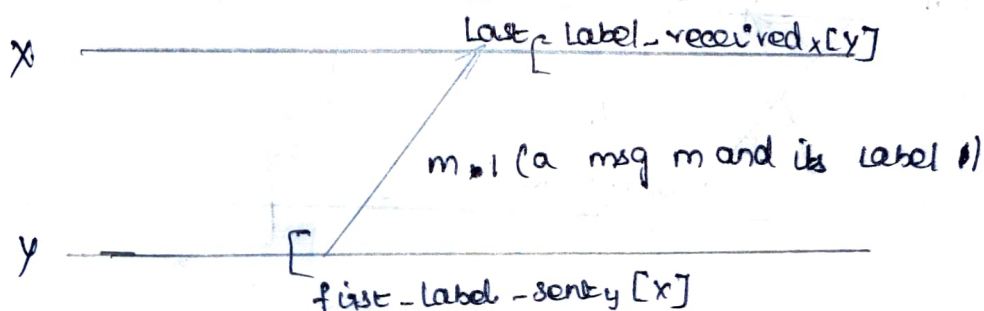


Consistent set of checkpoints.

Set $\{x_1, y_1, z_1\}$ is strongly consistent set of checkpoints.

Set $\{x_2, y_2, z_2\}$ is consistent set of checkpoints.

No local checkpoint includes another process.



Synchronous checkpointing and recovery:-

Livelock problem during recovery is avoided by taking a consistent set of checkpoints.

checkpointing Algorithm:

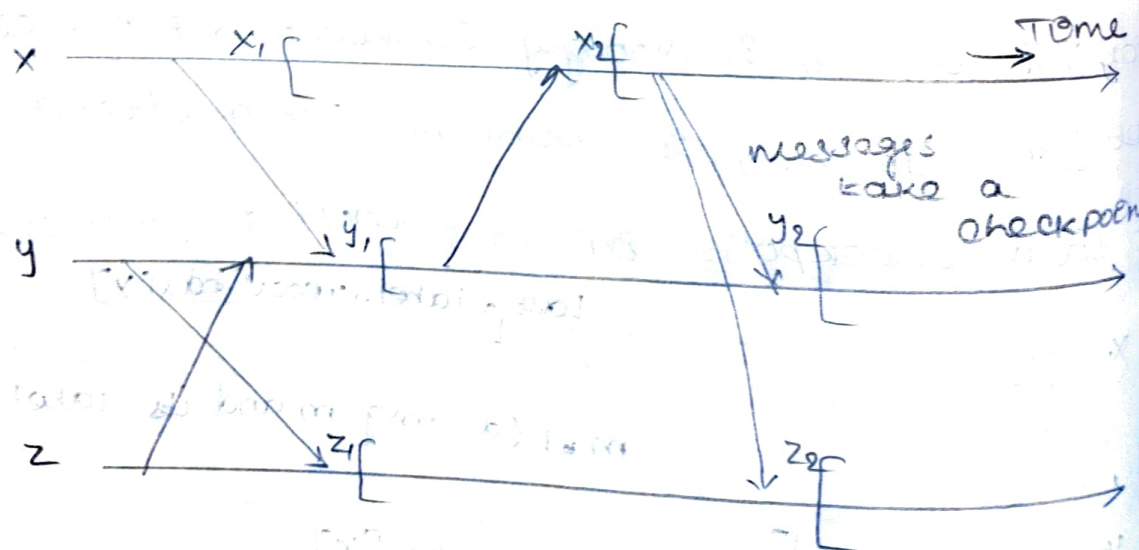
- ① Processes communicate by exchanging msg through channels
- ② Channels are FIFO, end-to-end protocol with msg boxes due to rollback recovery
- ③ Communication failure do not pause the n/w.

Two types of checkpoints:

① Tentative: A temporary checkpoint that is made a permanent checkpoint on the successful termination of the checkpoint algorithm.

② permanent: A local checkpoint at a process.

Optimization of the checkpoint Algorithm:

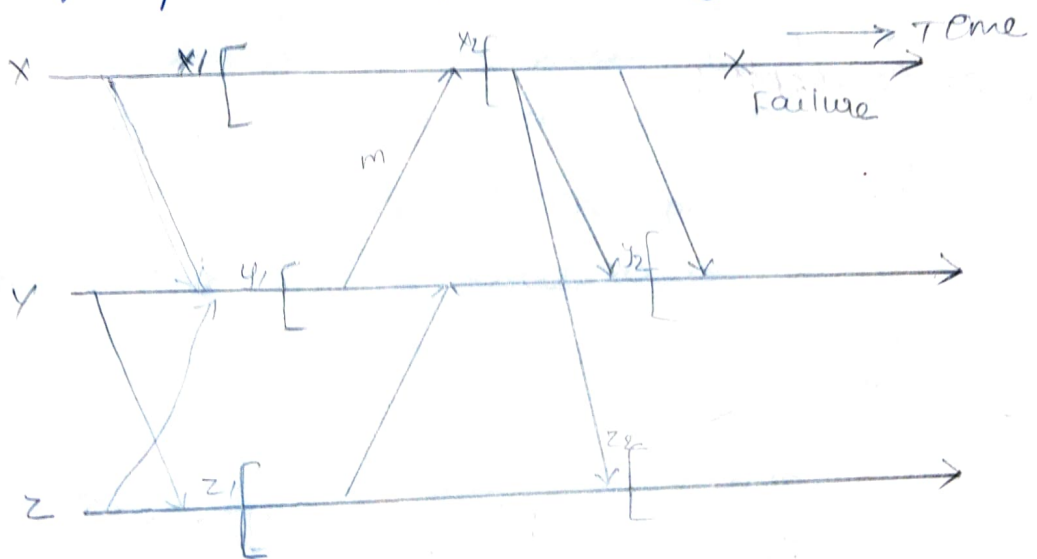


- (*) A minimal number of process take checkpoints.
- (*) All processes from which P₃ has received msg after it has taken its last checkpoint take a checkpoint to record the sending those msg.

The Rollback recovery Algorithm:

Optimization:

A minimum number of process roll back y will restore from its permanent checkpoint only if x is rolling back to a state where the sending of one or more msg from x to y is being undone.



Message Types:

- ① In transit msg
- ② Lost msg
- ③ Delayed msg
- ④ Orphan msg
- ⑤ Duplicate msg.

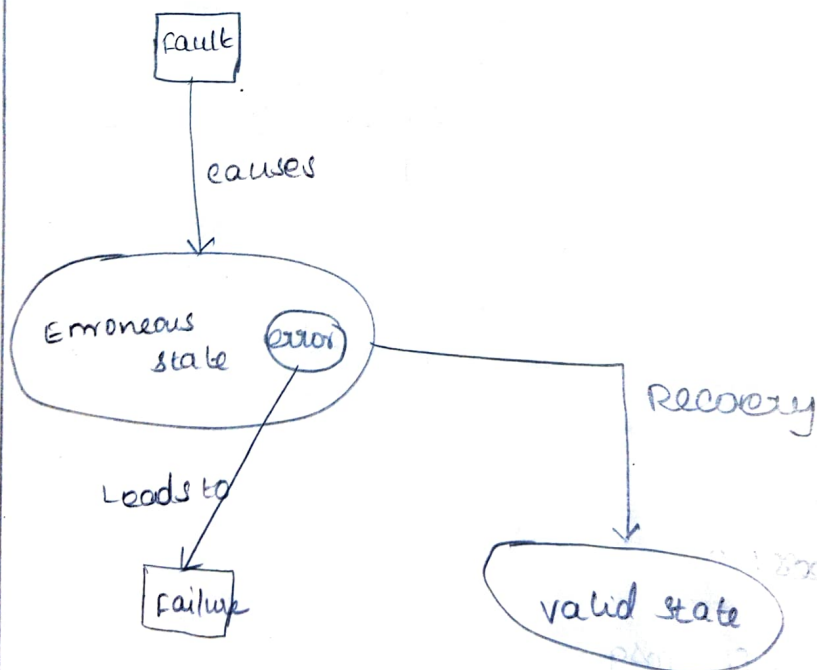
Issues in failure Recovery: - A/M - 2022

Recovery refers to restoring a s/m to its operational state. Once a failure has occurred, it is essential that the process where the failure has occurred to a correct state.

Fundamental to fault tolerance is the recovery from an error.

- ① Reclaim resource allocated to process
- ② Undo modification made to process
- ③ Restart the process
- ④ Or restart process from point of failure and resume execution.

Basic Concepts:



System failure: s/m does not meet requirements,
(i.e) does not perform its services as specified

Erroneous s/m state:

State which could lead to a s/m failure by a sequence of valid state transitions.

Error: The part of the s/m state which differs from its intended value.

Fault: Anomalous physical condition, e.g. design errors, manufacturing problems, damage, external disturbances.

Checkpoint-based Recovery:

The basic idea behind checkpoint-recovery is the saving & restoration of s/m state.

By saving the current state of the s/m periodically or before critical-code sections.

When a s/m is checkpointed, the state of the entire s/m is saved to non-volatile storage.

The checkpointing mechanism takes a snapshot of the s/m state and stores the data on some non-volatile storage medium.

① Uncoordinated checkpointing:

Each process has autonomy in deciding when to take checkpoints.

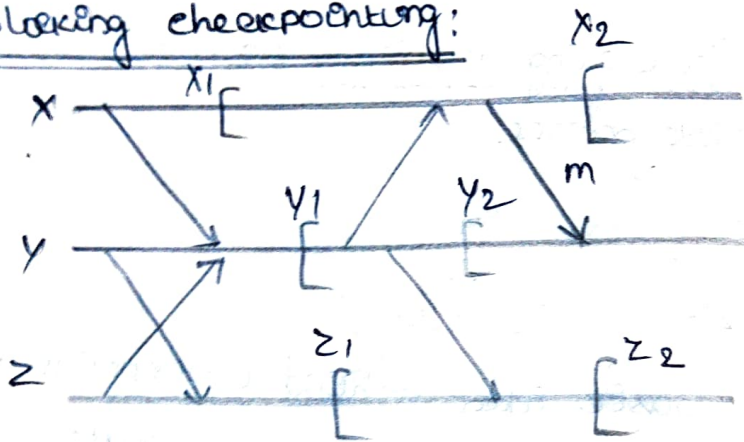
Adv: Lowers runtime overhead during normal execution.

Disadv: Domino effect if having a recovery.
Not suitable for application.

2) Coordinated checkpointing:

(*) Coordinated checkpointing simplifies failure recovery and eliminates domino effects in case of failures by preserving a consistent global checkpoint on stable storage.

Blocking checkpointing:



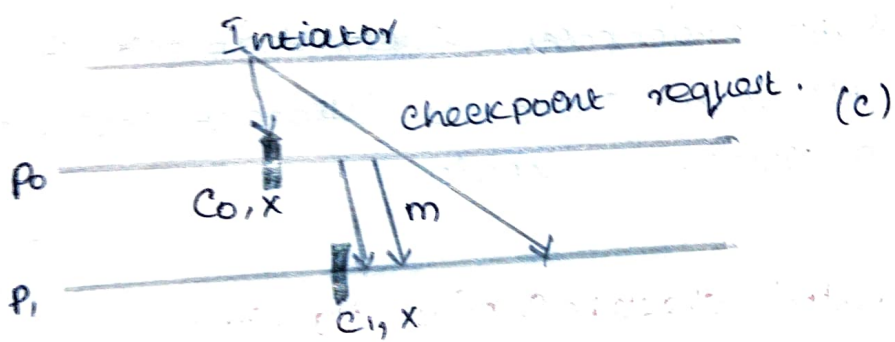
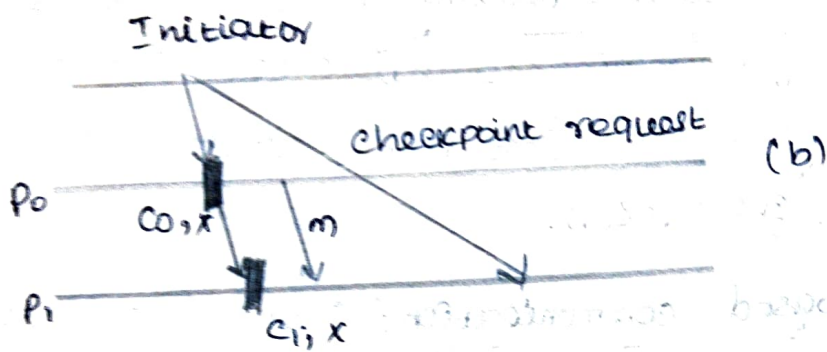
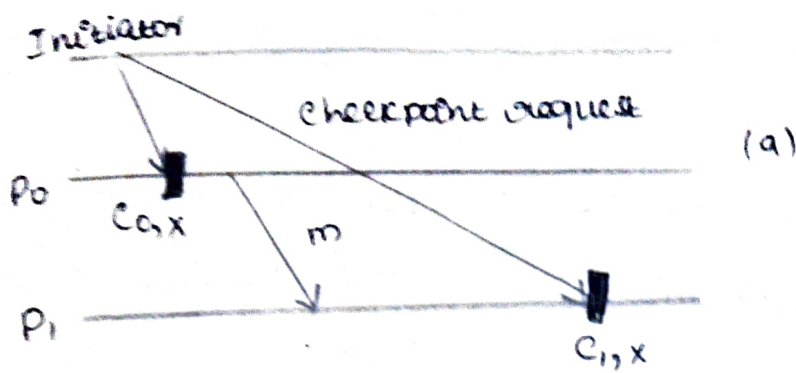
After a process takes a local checkpoint, to prevent orphan messages, it remains blocked until the entire checkpointing activity is complete.

Disadv: The computation is blocked during the checkpointing.

Non-blocking checkpointing:

(*) The processes need not stop their execution while taking checkpoints.

(*) A fundamental problem in coordinated checkpointing is to prevent a process from receiving application messages that could make the checkpoint inconsistent.



3) Communication - Induced checkpointing:

→ CIC protocols are popular, because they help in bounding rollback propagation during failure recovery, by ensuring that each checkpoint taken is part of a consistent global checkpoint of the distributed computation.

→ It avoids domino effect, while allowing processes to take some of their checkpoints independently.

Communication induced
piggybacked protocol and related
msg.

The receiver of each application msg
the piggybacked protocol and related msg
on each application message.

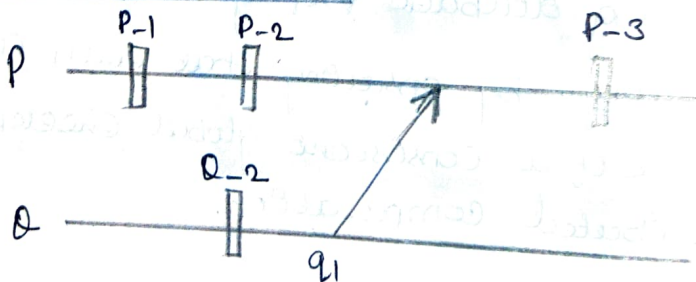
Model-based checkpointing relies on previous
patterns of communication and checkpoints that
could result in inconsistent states among
existing checkpoints.

Index-based communication: Induced check-
points by assigning monotonically increasing
indices, to checkpoints, such that the check-
points having the same index at different processes
form a consistent state.

Coordinated checkpointing Algorithm

*) Uncoordinated checkpointing may lead to
domino effect or to live-lock.

Koo-Toueg Algorithm:



Koo-Toueg In 1981 proposed a coordinated
checkpointing and recovery technique that
takes a consistent set of checkpoints and avoids
domino effect and live-lock problems during

(*) Suppose P wants to establish a checkpoint at $P-3$. This will record that q_1 was received from Q , to prevent q_1 from being orphaned, Q must checkpoint as well.

(*) Thus, establishing a checkpoint at $P-3$ by P forces Q to take a checkpoint to record that q_1 was event.

(*) An algorithm for ~~an~~ such coordinated checkpointing has two types of checkpoints - tentative and permanent.

(*) Call the set of such process π .

(*) The message tells each process in π the last message, $m_{q,p}$, that P has received from q before the tentative checkpoint was taken.

(*) This may set off a chain reaction of checkpointing.

(*) Each member of π can potentially spawn a set of checkpoints among processes in its corresponding set.

Algorithm for Asynchronous Checkpointing and Recovery

Jang - Venkatesan algorithm for asynchronous checkpointing and recovery.

Assumptions: Communication channels are reliable, delivery msg in FIFO order, infinite buffer, msg transmission delay is arbitrary but finite.

In this recovery algorithm each process keeps track of both the number of messages it has sent to and received from the other processes.

Whenever a process rolls back, it is necessary for all other processes to find if any message sent by the rolled back process has become an orphan message.

Two types of log storage are maintained:

a) Volatile log: short time to access but lost if processor crash. Move to stable log periodically.

b) Stable log: longer time to access but remained if crashed.

Asynchronous checkpointing: After executing an event, the tuple is recorded without any synchronization with other processes. Local checkpoint consists of set of records first are stored in volatile log, then moved to stable log.

Recovery algorithm:

(*) Number of messages received by P_i , from P_j , from the beginning of computation to checkpoint.

(*) Number of messages sent by P_i to P_j from beginning of computation to checkpoint.

(*) Idea: From the set of checkpoints, find a set of consistent checkpoints. Doing that based on the number of messages sent and received.

UNIT V

5 CLOUD COMPUTING

1. Definition of cloud computing :-

cloud computing is a general term used to describe a new class of network based computing that take place over the Internet, basically a step on from utility computing.

cloud components :-

Client: mobile terminal (or) regular computer

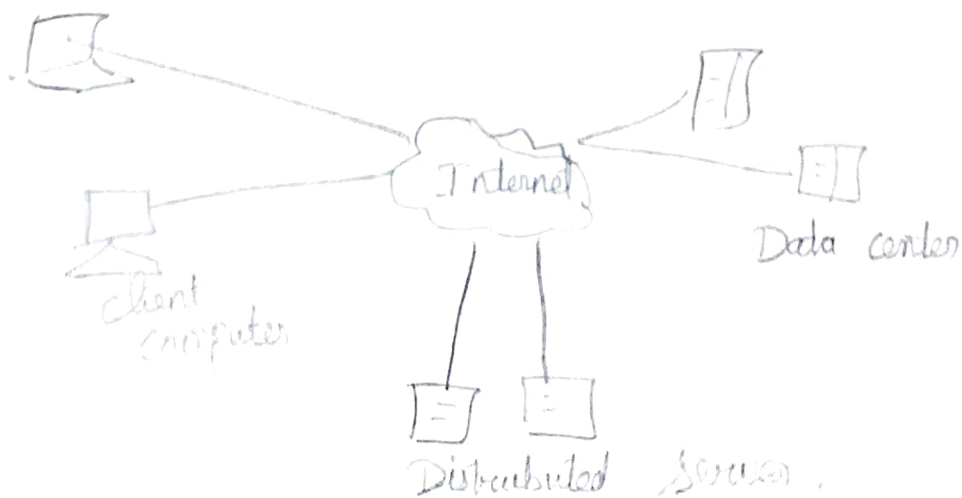
Benefit :- lower hardware cost, lower IT cost, security

data security, less Power consumption

Data center: Collection of server where the application to subscribe is housed.

Virtualizing server :- Software can be installed allowing multiple instances of virtual server.

Distributed server: Server don't all have to be housed in the same location.



Pros and cons of cloud computing :-

Pros of cloud computing :-

1. Lower computer set
2. Improved performance
3. Reduced software costs
4. Instant software update

Cons of cloud computing

1. It requires a constant Internet connection
2. Features might be limited
3. Stored data might not be secure.
4. Does not work well with low-speed connection

5.1.3 Application of cloud computing :-

1. Through cloud cost flexibility, online market place gains access to more powerful analytics online
2. Greater business
3. Greatest market adaptability provides online entertainment
4. Masked complexity enables access to services

5.2 Characteristics of cloud :-

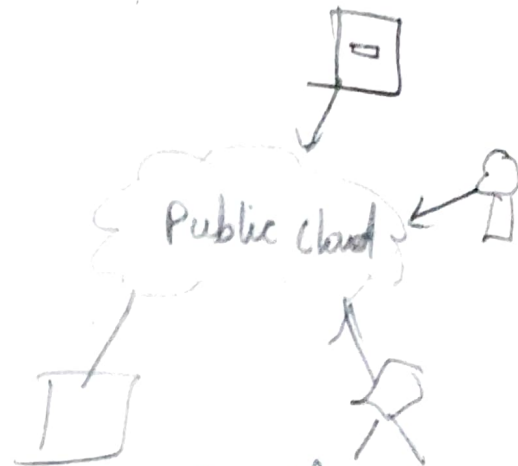
1. on demand self service
2. ubiquitous network access
3. location independent resource pooling
4. Rapid elasticity
5. Pay per use

5.3 cloud deployment models:-

* cloud deployment models are refer to the location and management of cloud infrastructure



a) Private cloud



b) Public cloud

1) Public cloud :-

- * The cloud infrastructure is made available to the general public.
- * Public cloud is huge data center that offer the same services to all its users.
Eg: Facebook, Google.

2. Private cloud :-

- * The cloud infrastructure is operated solely for a single organization. It may be managed by the organization.

3) community cloud :-

- * The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns.

Hybrid cloud :-

The cloud infrastructure is a composition of two (or) more clouds that remain unique entities but are bound together by standardized (or) proprietary technology.

Hybrid cloud benefits :-

- a) Hybrid clouds are still being developed, not many in real use.

5.4) cloud service models :-

* Service models describe the type of service that the service provider is offering - the best-known service models are software as a service, platform as a service and Infrastructure as a service

*

Cloud clients
(web browser, mobile app, thin client, terminal emulator, ...)



SaaS

CRM, Email, virtual desktop, communication games

PaaS

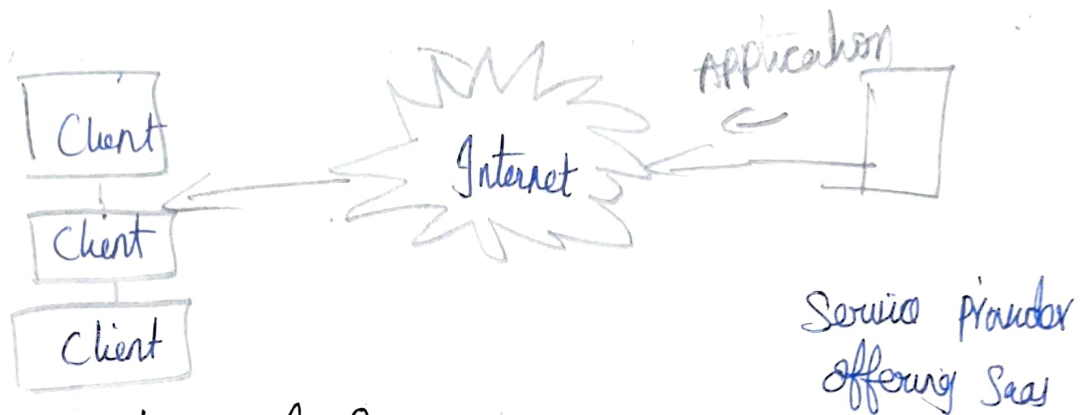
Execution runtime, database, web services, development tool

IaaS

Virtual machine, server, storage, load balancer.

5.4.1 Software as a Service :-

- * model in which an application is hosted as a service to customers who access it via the Internet
- * The provider does all the patching and upgrades as well as keeping the infrastructure running



Characteristics of SaaS :-

- * Software applications (or) services are stored remotely
- * A user can then access these service (or) software application via internet

Benefits of SaaS :-

- * Easier administration and invoicing

5.4.2 Platform as a service :-

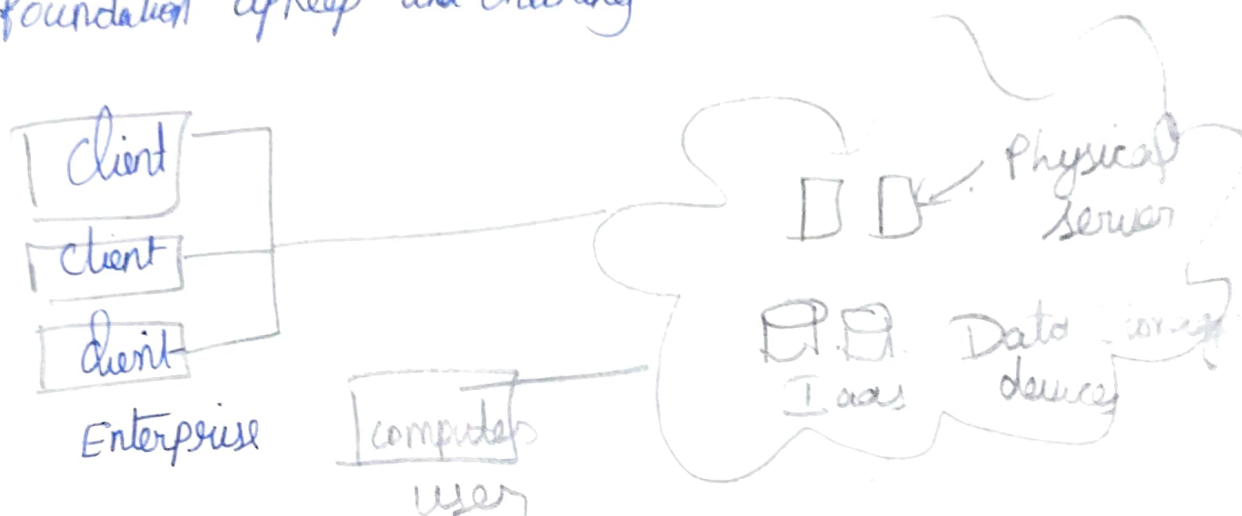
- * Platform as a service is another application delivery model and also known as cloud-ware
- * service include : Application design, development, testing, deployment and hosting

Characteristics of PaaS :-

- * It support multi-tenanted architecture
- * It support for development of group collaboration

5.4.3. Infrastructure as a service (IaaS)

- * IaaS gives the storage room likeness to the in-house datacenter stood out from various organization sorts
- * center datacenter framework, segments core capacity, serves the system itself and administration apparatuses for foundation upkeep and checking



Driving factors and challenges of cloud :-

- * Increased security vulnerabilities
- * Reduced operational governance control
- * limited portability between cloud provider.

5.4 Virtualization :-

- * Virtualization is a broad term that refers to the abstraction of resources across many aspects of computing
- * virtualization is an abstraction layer that decouples the physical hardware from the operating system to deliver greater IT resource utilization and flexibility

5.6.1 Hypervisor :-

- * A hypervisor is a software layer installed on the physical hardware, which allows splitting the physical machine into many virtual machines. These allow multiple operating systems to be run simultaneously on the same physical hardware.

5.6.2 Para Virtualization :-

- * Para virtualization is a type of virtualization in which a guest operating system (OS) is recompiled, installed inside a virtual machine (VM) and operated on top of hypervisor program running on the host OS.

5.6.3 Full -virtualization :-

- * Full virtualization doesn't need to modify the host OS. It relies upon binary translation to trap and to virtualize certain sensitive instructions.

5.7 Load Balancing :-

- * Load balancing can be defined as the process of task distribution among multiple computers, process, disk (or) other resources in order to get optimal resources utilization and reduce the computation time.
- * Google, Yahoo. million of user hits per day. Across the web, sites experience a wide range of network traffic requirement.

there are ~~to~~ three types

- 1) centralized approach
- 2) Distributed approach
- 3) mixed approach.

5.8 Scalability and Elasticity :-

Scalability is the ability of a system (or) network to handle increased load (or) usage. At the same time, elasticity is the ability to automatically expand and contract resources to meet demand.

Scalability is one of the prominent features of cloud computing. In the past, a system scalability relied on the company hardware and thus, was severely limited in resources.

5.9 Replication :-

* cloud replication refers to the process of replicating data from on-premises storage to the cloud. Traditional data replication involves replicating data across different physical servers on the computer local network.

5.10 monitoring :-

* monitoring service allow cloud user to collect and analyse the data on various monitoring metrics. A monitoring service collect data on various system and application metrics from the cloud computing instances.

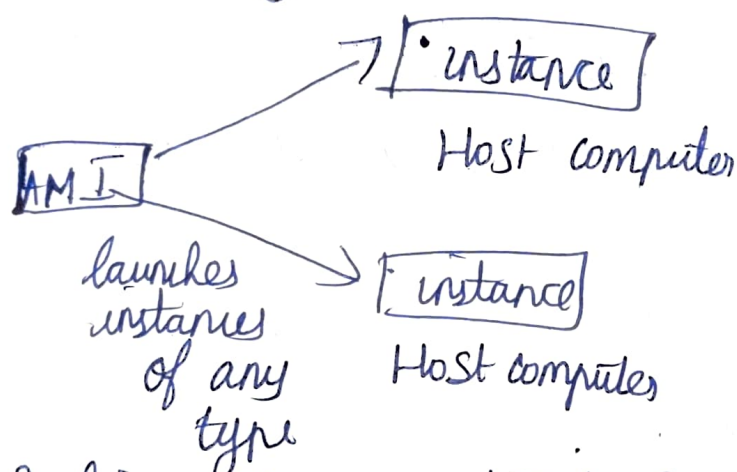
5.11 Cloud Services and Platform: compute services

* compute service contains the fundamental element of cloud computing system

5.11.1 Amazon Elastic compute cloud:-

* Amazon Elastic compute cloud is a web service that provides resizable compute capacity in the cloud

It is designed to make webScale computing easier for developers and system administrators.



* Auto Scaling allow automatically scale of the capacity up seamlessly during demand spikes to maintain performance and scales down during demand lulls and minimize cost

5.11.2 Windows Azure:-

* Windows Azure is a cloud computing platform and infrastructure created by Microsoft for building, deploying and managing application services through a global network of Microsoft-managed data centers

5.12 Storage service:-

Amazon S3 defines a bucket name as a series of one (or) more labels separated by period. The address follows the following rule: The bucket name can be between 3 and 63 character long.

Google cloud storage:-

* It allows world-wide storage and retrieval of any amount of data at any time.

- a) cloud console
- b) client libraries
- c) gcloud
- d) Rest API

5.13 Application services:-

Google App engine:-

* It is a service cloud computing platform for developing and hosting web applications in google-managed data center.